CHAPTER - 1
INTRODUCTION

In the era of digital connectivity, web applications have become indispensable tools for businesses, organizations, and individuals alike. These applications handle a vast amount of sensitive data, from financial transactions to personal information, making them prime targets for cyberattacks. Web vulnerabilities, which are weaknesses in web applications that can be exploited by attackers, pose a significant threat to the security and integrity of these applications.

1.1 Overview

Web applications are the most common interface to security sensitive data and functionality available nowadays. They are routinely used to file tax incomes, access the results of medical screenings, perform financial transactions, and share opinions with our circle of friends, just to mention a few popular use cases. On the downside, this means that web applications are appealing targets to malicious users (attackers) who are determined to force economic losses, unduly access confidential data or create embarrassment to their victims. Securing web applications is well known to be hard.

There are several reasons for this, ranging from the heterogeneity and complexity of the web platform to the adoption of undisciplined scripting languages offering dubious security guarantees and not amenable for static analysis. In such a setting, black-box vulnerability detection methods are particularly popular. As opposed to white-box techniques which require access to the web application source code, black-box methods operate at the level of HTTP traffic, i.e., HTTP requests and responses. Though this limited perspective might miss important insights, it has the key advantage of offering a language-agnostic vulnerability detection approach, which abstracts from the complexity of scripting languages and offers a uniform interface to the widest possible range of web applications.

1.2 Scope

This project delves into the realm of Machine Learning (ML) and its application in combating Cross-Site Request Forgery (CSRF), a prevalent web vulnerability. It begins with an introduction to CSRF, unraveling its attack mechanism and potential consequences for web applications. Subsequently, it delves into ML-based CSRF detection strategies, exploring techniques such as pattern recognition, anomaly detection, and supervised learning algorithms. The project highlights the advantages of ML-based CSRF detection systems over traditional signature-based methods, emphasizing their ability to identify novel and unknown attacks, adapt to evolving web applications, and provide contextual information.

This work provides the most up to date and comprehensive account of the nature of the CSRF attacks and corresponding solution to thwart these attacks. However, as the new knowledge pioneered in this work takes hold, the future extensions of this knowledge are bound to happen. Some areas of further work, in future, are identified as follows: As a future extension of this work it may be possible to perform the Bayesian estimation by using 1-99 (configurable) threshold probability ratio of suspect CSRF page to safe pages since anything lower than 1% of the threshold can either be random or an unexpected variance incorporating routine CSRF scanning in commercial anti-malware. Browser specific and generic anti CSRF solutions. This Machine Learning (ML) Web Vulnerability Detection project has a broad scope aimed at significantly advancing web application security. The primary focus is on automating the identification of vulnerabilities, including but not limited to common threats like Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL injection. The project acknowledges the diversity of web applications, accommodating their varying complexities, structures, and programming practices. Its adaptability to custom programming practices ensures a robust solution across a wide spectrum of applications. Operating as a black-box vulnerability detection system, the project's versatility lies in its capability to analyze HTTP traffic without requiring access to the source code. A critical aspect of the project's scope involves continuous monitoring and updates to address emerging vulnerabilities, ensuring the model's relevance, compatibility, capability and effectiveness over a long time period.

1.3 Definitions

S. No	Name	Definitions
1	Machine Learning (ML)	A field of computer science that enables systems to learn and improve from experience without being explicitly programmed. ML algorithms can identify patterns and make predictions based on data, making them well-suited for tasks such as web vulnerability detection.
2	Cross-Site Request Forgery (CSRF)	A type of web attack that tricks a user into submitting unwanted requests to a web application in which they are currently authenticated. This can be done by embedding malicious code in a website or sending a link to the user that, when clicked, submits a malicious request to the vulnerable website.
3	Pattern Recognition	A technique used in ML to identify patterns in data. In the context of CSRF detection, pattern recognition can be used to identify patterns in HTTP requests and responses that are indicative of CSRF attacks.
4	Anomaly Detection	A technique used in ML to identify data points that deviate significantly from the norm. In the context of CSRF detection, anomaly detection can be used to identify HTTP requests that are unusual or unexpected and may be indicative of a CSRF attack.
5	Supervised Learning	A type of ML algorithm that learns from labeled data. In the context of CSRF detection, supervised learning algorithms can be trained on datasets of HTTP requests and responses that have been labeled as either benign or malicious.
6	False Positives	In the context of CSRF detection, a false positive occurs when an ML algorithm identifies a benign request as malicious.
7	Interpretability	Interpretability refers to the ability to understand how an ML algorithm makes its decisions. Interpretability is important for debugging false positives and improving the accuracy of ML-based CSRF detection systems.

Table – 1.1: Definitions

1.4 User Needs

Machine learning can be used for the detection of web application vulnerabilities. In order to leverage machine learning for web vulnerability detection, it is important to have complete data that includes all relevant details to avoid biasing the detection result or compromising the accuracy of the model. Some key considerations for using machine learning for web vulnerability detection include:

1. **Data completeness**: Missing certain details may make it difficult for a classifier to learn representative vulnerability features.

2. **Detecting undiscovered vulnerabilities**: Previously undiscovered vulnerabilities can be triggered by specific events or conditions, which makes it challenging to predict.

3. **Low rate of missed detection and false alarm**: Experimental results show that the model has a low rate of missed detection and false alarm, and the improved model is more efficient.

4. **Verification code identification function**: Based on the existing network vulnerability detection technology and tools, the verification code identification function is added, which solves the problem that the data can be submitted to the server only by inputting the verification code.

5. **Cross-site scripting security vulnerability detection model**: A cross-site scripting security vulnerability detection model for web application is designed and implemented.

1.5 Objective

The main aim of the project is to determine the secured and non-secured websites over the internet and detect the accuracy of the secure and non-secure sites.

CHAPTER – 2
REQUIREMENT SPECIFICATION

The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether.

2.1 Software Requirements

➢ Operating System	–	Windows
➢ Programming Language	–	Python
➢ Database	–	MySQL, SQLite
➢ Platform	–	VS Code
➢ Frontend and Designing	–	HTML, CSS, JavaScript

2.2 Hardware Requirements

➢ Processor	–	Intel Core i5
➢ RAM	–	8GB
➢ Hard Disk	–	1TB
➢ Monitor	–	14’ Color monitor
➢ Mouse	–	Optical mouse

CHAPTER – 3
LITERATURE SURVEY

3.1 Surviving The Web: A Journey into Web Session Security (2011).

Authors: Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta in the year, April 2017.

The Web is the primary access point to on-line data and applications. It is extremely complex and variegate, as it integrates a multitude of dynamic contents by different parties to deliver the greatest possible user experience. This heterogeneity makes it very hard to effectively enforce security, since putting in place novel security mechanisms typically prevents existing websites from working correctly or negatively affects the user experience, which is generally regarded as unacceptable, given the massive user base of the Web However, this continuous quest for usability and backward compatibility had a subtle effect on web security research: designers of new defensive mechanisms have been extremely cautious and the large majority of their proposals consists of very local patches against very specific attacks.

This piecemeal evolution hindered a deep understanding of many subtle vulnerabilities and problems, as testified by the proliferation of different threat models against which different proposals have been evaluated, occasionally with quite diverse underlying assumptions. It is easy to get lost among the multitude of proposed solutions and almost impossible to understand the relative benefits and drawbacks of each single proposal without a full picture of the existing literature.

In this work, we take the delicate task of performing a systematic overview of a large class of common attacks targeting the current Web and the corresponding security solutions proposed so far.

3.2 Large-Scale Analysis & Detection of Authentication Cross-Site Request Forgeries (2012)

Authors: Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli.

Cross-Site Request Forgery (CSRF) attacks are one of the critical threats to web applications. In this paper, we focus on CSRF attacks targeting web sites' authentication and identity management functionalities. We will refer to them collectively as Authentication CSRF (Auth-CSRF in short). We started by collecting several Auth-CSRF attacks reported in the literature, then analyzed their underlying strategies and identified 7 security testing strategies that can help a manual tester uncover vulnerabilities enabling Auth-CSRF.

In order to check the effectiveness of our testing strategies and to estimate the incidence of Auth-CSRF, we conducted an experimental analysis considering 300 web sites belonging to 3 different rank ranges of the Alexa global top 1500. The results of our experiments are alarming: out of the 300 web sites we considered, 133 qualified for conducting our experiments and 90 of these suffered from at least one vulnerability enabling Auth-CSRF (i.e. 68%).

We further generalized our testing strategies, enhanced them with the knowledge we acquired during our experiments and implemented them as an extension (namely CSRF-checker) to the open-source penetration testing tool OWASP ZAP. With the help of CSRF checker, we tested 132 additional web sites (again from the Alexa global top 1500) and identified 95 vulnerable ones (i.e. 72%).

Our findings include serious vulnerabilities among the web sites of Microsoft, Google, eBay etc. Finally, we responsibly disclosed our findings to the affected vendors.

3.3 State of the Art: Automated Blackbox Web Application Vulnerability Testing (2013)

Authors: Jason Bau, Elie Bursztein, Divij Gupta, and John C. Mitchell

Black-box web application vulnerability scanners are automated tools that probe web applications for security vulnerabilities. In order to assess the current state of the art, we obtained access to eight leading tools and carried out a study of:

(i) The class of vulnerabilities tested by these scanners.

(ii) Their effectiveness against target vulnerabilities.

(iii) The relevance of the target vulnerabilities to vulnerabilities found in the wild.

To conduct our study we used a custom web application vulnerable to known and projected vulnerabilities, and previous versions of widely used web applications containing known vulnerabilities. Our results show the promise and effectiveness of automated tools, as a group, and also some limitations.

In particular, "stored" forms of Cross Site Scripting (XSS) and SQL Injection (SQLI) vulnerabilities are not currently found by many tools.

Because our goal is to assess the potential of future research, not to evaluate specific vendors, we do not report comparative data or make any recommendations about purchase of specific tools.

This piecemeal evolution hindered a deep understanding of many subtle vulnerabilities and problems, as testified by the proliferation of different threat models against which different proposals have been evaluated, occasionally with quite diverse underlying assumptions. It is easy to get lost among the multitude of proposed solutions and almost impossible to understand the relative benefits and drawbacks of each single proposal without a full picture of the existing literature. In this work, we take the delicate task of performing a systematic overview of a large class of common attacks targeting the current Web and the corresponding security solutions proposed so far.

We focus on attacks against web sessions, i.e., attacks which target honest web browser users establishing an authenticated session with a trusted web application. This kind of attacks exploits the intrinsic complexity of the Web by tampering, e.g., with dynamic contents, client-side storage or cross-domain links, so as to corrupt the browser activity and/or network communication.

Our choice is motivated by the fact that attacks against web sessions cover a very relevant subset of serious web security incidents and many different defenses, operating at different levels, have been proposed to prevent these attacks.

Black-box scanners mimic external attacks from hackers, provide cost-effective methods for detecting a range of important vulnerabilities, and may configure and test defenses such as web application firewalls. Since the usefulness of black-box web scanners is directly related to their ability to detect vulnerabilities of interest to web developers, we undertook a study to determine the effectiveness of leading tools.

Our goal in this paper is to report test results and identify the strengths of current tools, their limitations, and strategic directions for future research on web application scanning method.

3.4 Why johnny can’t pentest: An analysis of black-box web vulnerability scanners (2017)

Authors: Adam Doup´e, Marco Cova, and Giovanni Vigna in the year, April 2017.

Black-box web vulnerability scanners are a class of tools that can be used to identify security issues in web applications. These tools are often marketed as “point-and-click pentesting” tools that automatically evaluate the security of web applications with little or no human support. These tools access a web application in the same way users do, and, therefore, have the advantage of being independent of the particular technology used to implement the web application.

However, these tools need to be able to access and test the application’s various components, which are often hidden behind forms, JavaScript-generated links, and Flash applications. This paper presents an evaluation of eleven black-box web vulnerability scanners, both commercial and open-source. The evaluation composes different types of vulnerabilities with different challenges to the crawling capabilities of the tools.

These tests are integrated in a realistic web application. The results of the evaluation show that crawling is a task that is as critical and challenging to the overall ability to detect vulnerabilities as the vulnerability detection techniques themselves, and that many classes of vulnerabilities are completely overlooked by these tools, and thus research is required to improve the automated detection of these flaws.

CHAPTER - 4
SOFTWARE REQUIREMENT ANALYSIS

4.1 Problem Definition

Cross-Site Request Forgery (CSRF) is a well-known web attack that forces a user into submitting unwanted, attacker controlled HTTP requests towards a vulnerable web application in which she is currently authenticated. The key concept of CSRF is that the malicious requests are routed to the web application through the user’s browser, hence they might be indistinguishable from intended benign requests which were actually authorized by the user. The CSRF does not require the attacker to intercept or modify user’s requests and responses: it suffices that the victim visits the attacker’s website, from which the attack is launched. Thus, CSRF vulnerabilities are exploitable by any malicious website on the web. Cross-Site Request Forgery represents a well-established and pervasive web security threat, wherein malicious actors compel users to unknowingly execute undesired, attacker-controlled HTTP requests within a susceptible web application to which the user is authenticated. The crux of CSRF lies in the surreptitious routing of these malevolent requests through the user's own browser, rendering them virtually indistinguishable from legitimate and authorized requests initiated by the user. What distinguishes CSRF from other web attacks is its intrinsic capacity to operate without necessitating the interception or modification of the user's requests and responses. In essence, the attacker's exploitation of CSRF is not contingent on direct manipulation of user communications. Instead, all that is required is for the unsuspecting victim to visit a website initiated by the attacker, from which the CSRF attack is executed. Consequently, the ubiquity of CSRF vulnerabilities exposes web applications to exploitation by any malevolent site on the internet, emphasizing the critical need for robust countermeasures to safeguard against this pervasive and potentially damaging threat.

4.2 Functional Requirements

A functional requirement defines a function of a system or its component, where a function is described as a specification of behavior between inputs and outputs. Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionalities that define what a system is supposed to accomplish. These are the requirements that the user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed, and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. Behavioral requirements describe all the cases where the system uses the functional requirements, these are captured in use cases. Functional requirements are supported by non-functional requirements also known as "quality requirements", which impose constraints on the design or implementation such as performance requirements, security, or reliability.

1. Data Collection
2. Data Preprocessing
3. Training and Testing
4. Modeling
5. Predicting

4.3 Non – Functional Requirements

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability, and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, “how fast does the website load?” Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non- functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile 26 backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recover ability requirement.
- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

CHAPTER - 5

SOFTWARE DESIGN

5.1 Architectural Design

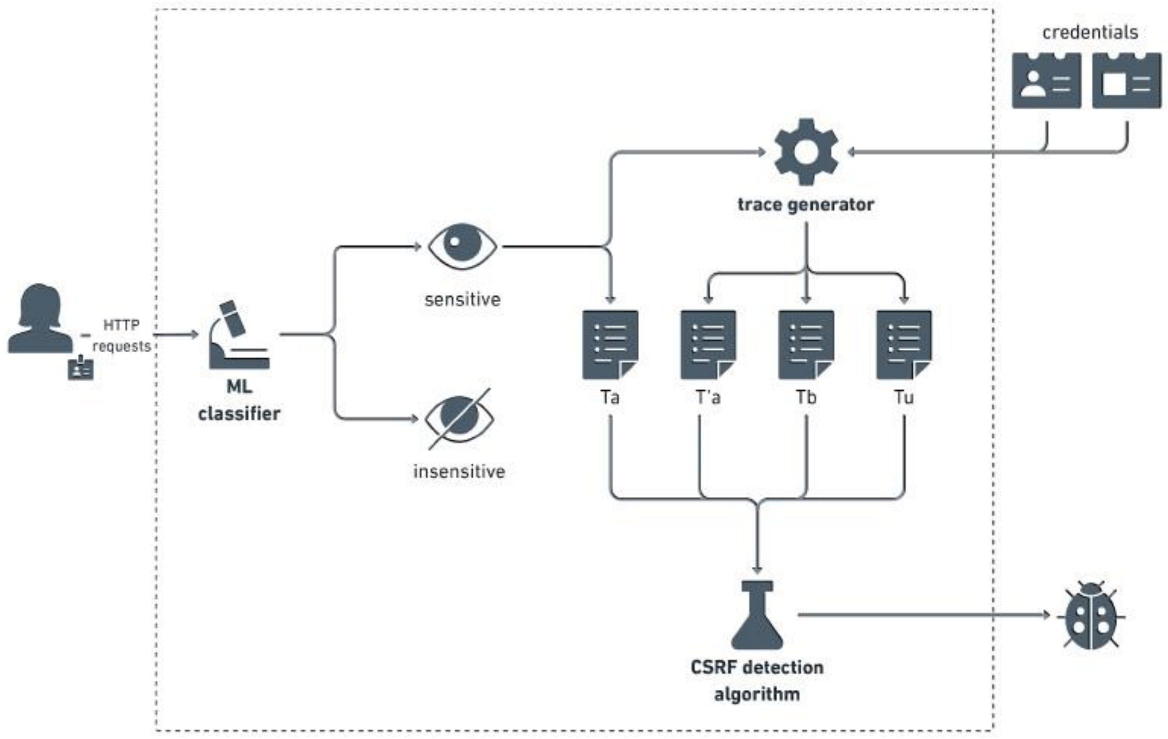


Fig – 5.1.1 Architectural Design

The process begins with capturing HTTP requests and responses generated by web applications. This data is then preprocessed to remove sensitive information and extract relevant features, such as timestamps, URLs, request bodies, and user-related information. These features are then fed into an ML classifier, which has been trained to distinguish between benign and malicious HTTP requests. The classifier's output, combined with the extracted features, is used by a CSRF detection algorithm to make a final determination about whether a request is legitimate or an attempt to exploit a Cross Site Request Forgery vulnerability (CSRF) that will lead to the output.

5.2 Modules

In our project, mainly, we use four types of Modules. They are

1. User
2. Admin
3. False Positives and False Negatives
4. Machine Learning Classifier

Let us understand about these four modules

1. User

The User can register the first. While registering he required a valid user email and mobile for further communications. Once the user register then admin can activate the customer. Once admin activated the customer then user can login into our system. User can do the data preprocess. First required running website name. By using that website the user can test the CSRFs. By help of bolt tool the user can fetch related all CSRFs and generated algorithm names. The result will be stored in JSON files. Later the user can get the results of Mitch dataset. The mitch dataset tested for POST method as well GET method to. The result will be displayed on the browser.

2. Admin

Admin can login with his credentials. Once he login he can activate the users. The activated user only login in our applications. The admin can set the training and testing data for the project of the Mitch Dataset. The user search all URLs related CSRF token admin can view in his page. The admin can also check the POST method performed data from the dataset and GET method related data also.

3. False Positives and False Negatives

Mitch produces a false positive when it returns a candidate CSRF that cannot be actually exploited. This is something relatively easy to detect by manual testing, though this process is tedious and time-consuming. In general, it is not possible to reliably identify when Mitch produces a false negative, because this would require to know all the CSRF vulnerabilities on the tested websites.

To estimate this important aspect, we keep track of all the sensitive requests returned by the ML classifier embedded into Mitch and we focus our manual testing on those cases. This is a reasonable choice to make the analysis tractable, because we first showed that the classifier performs well using standard validity measures.

4. Machine Learning Classifier

The ML classifier used by Mitch was trained from a dataset of around 6000 HTTP requests from existing websites, collected and labeled by two human experts. The feature space X of the classifier has 49 dimensions, each one capturing a specific property of HTTP requests. Those can be organized into following categories following set of numerical features:

- numOfParams: the total number of parameters;
- numOfBools: the number of request parameters bound to a boolean value;
- numOfIds: the number of request parameters bound to an identifier, i.e., a hexadecimal string, whose usage was empirically observed to be common in our dataset;
- numOfBlobs: the number of request parameters bound to a blob, i.e., any string which is not an identifier;
- reqLen: the total number of characters in the request, including parameter names and values.

5.3 UML Diagrams

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta- model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

1. Use-Case Diagram

A Use - Case Diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis.

This use case diagram illustrates the roles and interactions between users and admins within a system. Users can log in, access data, and submit requests, while admins have the additional authority to create, manage, and configure user accounts, as well as modify system settings. These interactions represent the core functionalities of the system, enabling users to perform tasks and admins to maintain control over the system's operation.

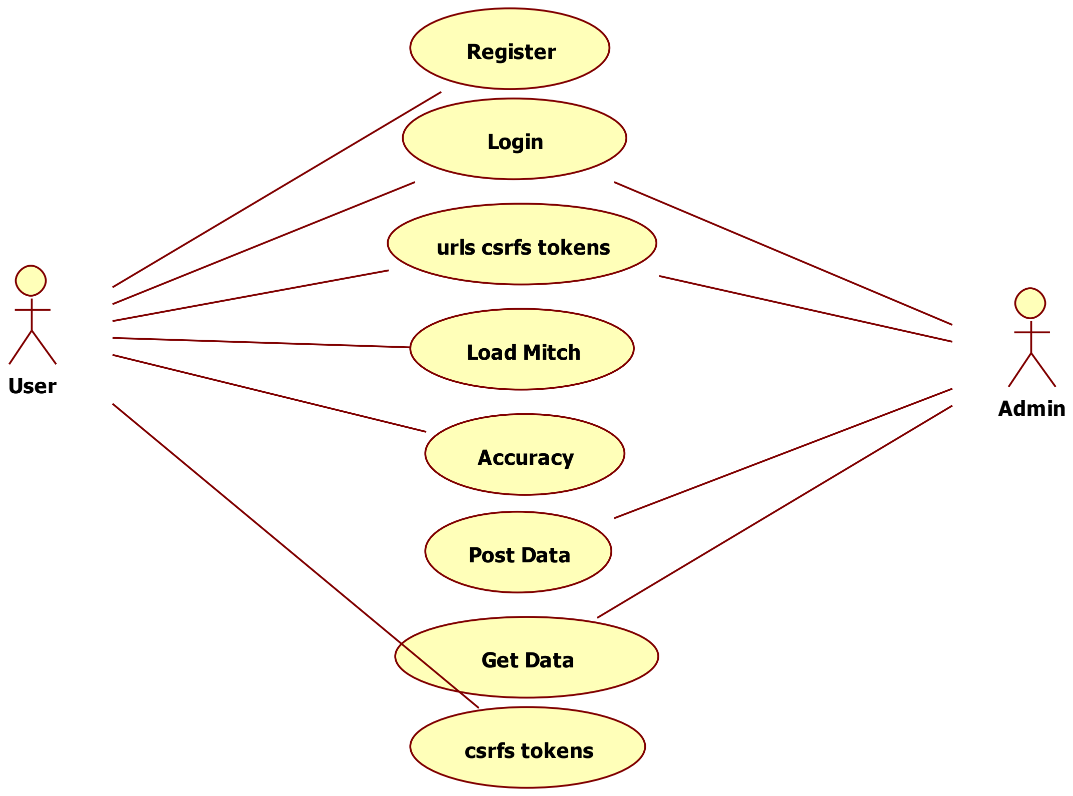


Fig – 5.3.1 Use-Case Diagram

2. Sequence Diagram

A Sequence Diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

This sequence diagram depicts the interactions between four modules: User, Admin, Bolt, and Mitch. It outlines the steps involved in processing a user's data request, highlighting the roles of different modules and the flow of information between them. The User initiates the request, the Admin approves it, Bolt processes it, Mitch provides the necessary data, Bolt updates the data, and the Admin informs the User of the outcome. This sequence diagram provides a clear overview of the system's data retrieval and processing workflow.

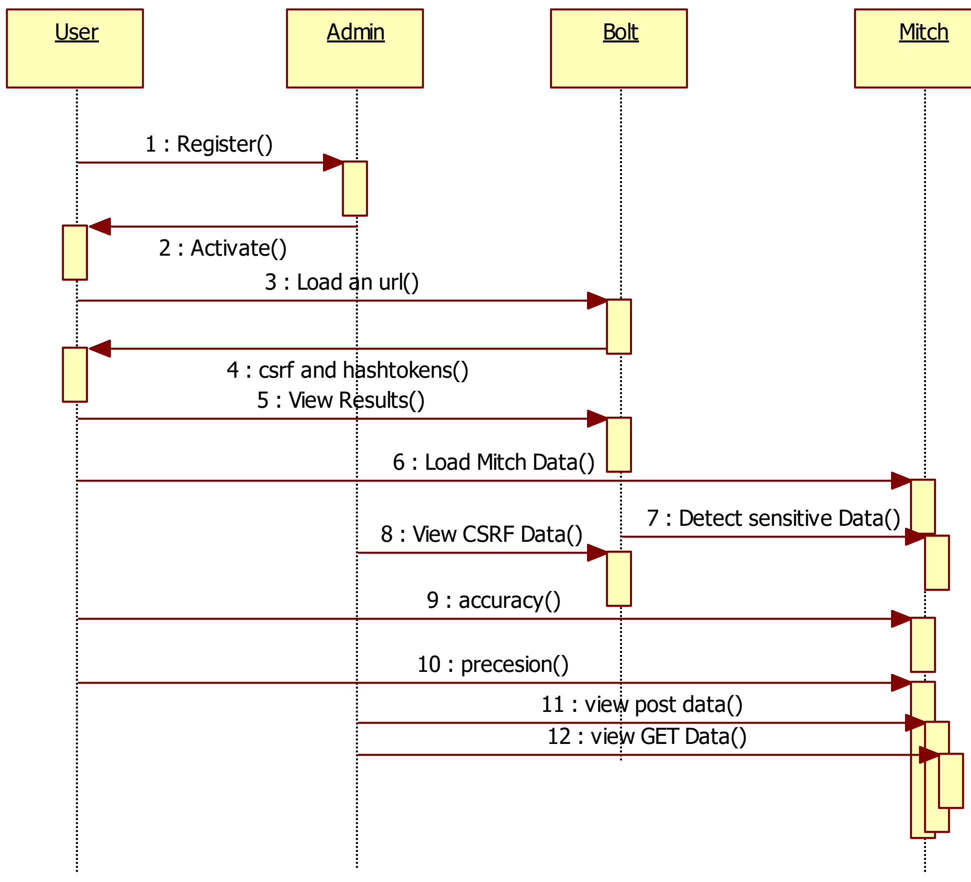


Fig – 5.3.2 Sequence Diagram

3. Class Diagram

This Class diagram illustrates the structure and interactions of a web application security system that utilizes machine learning for CSRF detection. The User interacts with the system, generating HTTP traffic that is processed by the CSRF Detection Module. The CSRF Detection Module employs the Feature Extractor to extract relevant features from the HTTP data and utilizes the ML Classifier to determine whether a request is benign or malicious. The HTTP Data Handler manages the collection, preprocessing, and storage of HTTP data. The Notification Module alerts relevant parties about detected CSRF attacks. This class diagram provides a clear representation of the system's architecture and the responsibilities of different modules in ensuring the security of the web application against CSRF attacks.

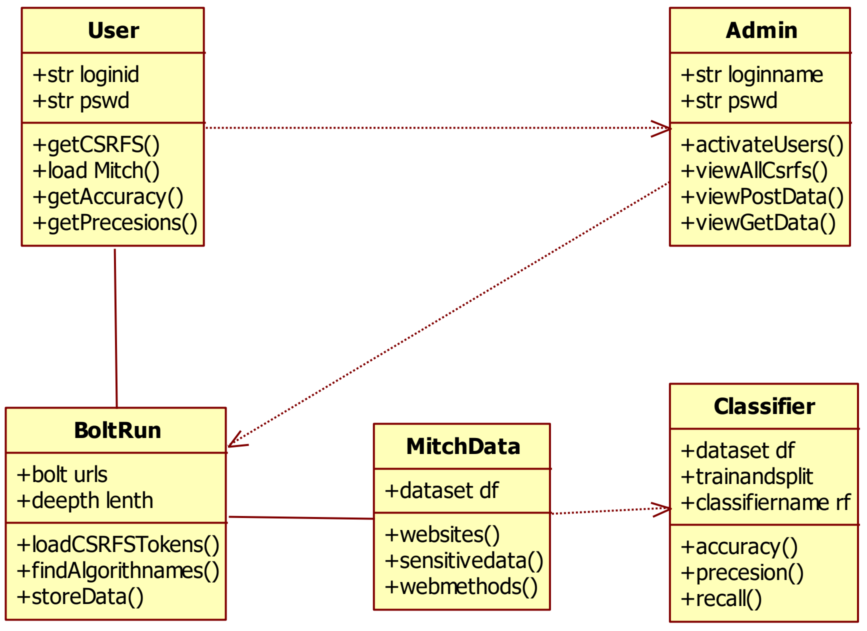


Fig – 5.3.3 Class Diagram

4. Activity Diagram

The activity diagram depicts the machine learning-driven process of identifying and mitigating Cross-Site Request Forgery (CSRF) attacks. It begins by capturing HTTP traffic, extracting relevant features, and training an ML classifier to distinguish between benign and malicious requests. The trained classifier is deployed into the web application security infrastructure, enabling real-time detection of potential CSRF attacks. Upon detection, the system triggers mitigation actions, such as invalidating session cookies or redirecting users, to prevent unauthorized access and maintain web application security.

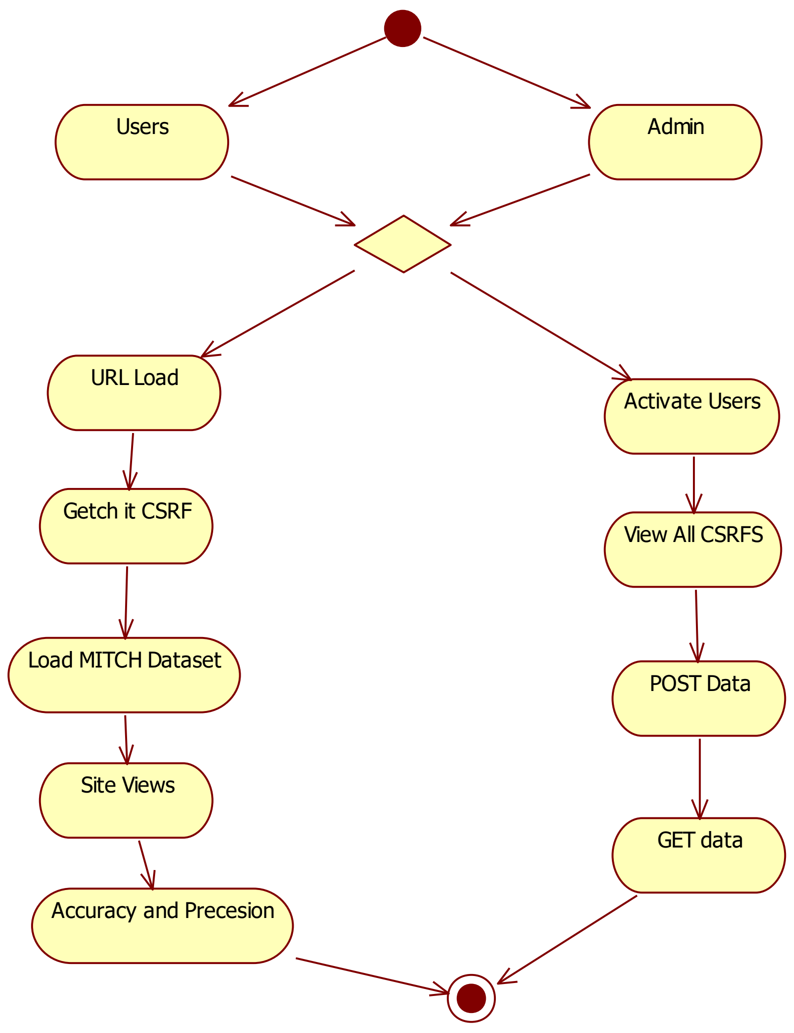


Fig – 5.3.4 Activity Diagram

CHAPTER - 6

SOURCE CODE

User Side views.py

```
from django.shortcuts import render, HttpResponse
from django.contrib import messages
from .forms import UserRegistrationForm
from .models import UserRegistrationModel, UserSearchUrlModel, CSRFResponse
import json
import subprocess
import pandas as pd
from .UserMachineLearningAlgorithms import MLConcepts


# Create your views here.

def UserRegisterActions(request):
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            print('Data is Valid')
            form.save()
            messages.success(request, 'You have been successfully 
registered')
            form = UserRegistrationForm()
            return render(request, 'UsersRegister.html', {'form': form})
        else:
            messages.success(request, 'Email or Mobile Already Existed')
            print("Invalid form")
    else:
        form = UserRegistrationForm()
    return render(request, 'UsersRegister.html', {'form': form})


def UserLoginCheck(request):
    if request.method == "POST":
        loginid = request.POST.get('loginname')
        pswd = request.POST.get('pswd')
        print("Login ID = ", loginid, ' Password = ', pswd)
        try:
            check = UserRegistrationModel.objects.get(loginid=loginid, 
password=pswd)
            status = check.status
            print('Status is = ', status)
            if status == "activated":
                request.session['id'] = check.id
                request.session['loggeduser'] = check.name
                request.session['loginid'] = loginid
                request.session['email'] = check.email
                print("User id At", check.id, status)
                return render(request, 'users/UserHome.html', {})
            else:
                messages.success(request, 'Your Account Not at activated')
                return render(request, 'UserLogin.html')
            # return render(request, 'user/userpage.html',{})
        except Exception as e:
            print('Exception is ', str(e))
            pass
        messages.success(request, 'Invalid Login id and password')
    return render(request, 'UserLogin.html', {})


def UserHome(request):
    return render(request, "users/UserHome.html", {})


def UserPreProcessForm(request):
    return render(request, "users/UserPreProcessForm.html", {})


def UserCSRFProcessByBolt(request):
    if request.method == "POST":
        urlname = request.POST.get("urlname")
        depth = request.POST.get("depth")
        UserSearchUrlModel.objects.create(urlname=urlname, depthfecth=depth)
        command = "python Bolt-master/bolt.py" + " -u " + urlname + " -l" + "
" + depth
        print("path " + command)
        subprocess.call(command)
        f = open('./db/hashes.json', )
        data = json.load(f)
        # print("Data is ",len(data))
        mydict = {}
        for i in data:
            keys = i.keys();
            # print("fu=",keys['regex'])
            # print("fus",keys['matches'])
            for x in keys:
                regex = i['regex']
                matches = i['matches']
                CSRFResponse.objects.create(regex=regex, matches=matches,
urlname=urlname)

    data = CSRFResponse.objects.filter(urlname=urlname)
    return render(request, "users/CSRFProcess.html", {"data": data})


def UserMitchProcess(request):
    f = open('./media/dataset/dataset.json', )
    data = json.load(f)
    mydict = {}
    for i in data:
        keys = i.keys()
        data = i['data']
        website = i['website']
        i = 0
        for x in data:
            #print("X value = ",x)
            i = i+1
            mydict.update({i:x})
        #mydict.update({data: website})
        #for x,y in keys.items():
            #data = keys.get('data')
            #website = i['website']
            #print(y)

            #print(data,"<==>",website)

    return render(request, "users/MitchProcessone.html", {"data": mydict})

def UserMachineLearning(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',',
delimiter=None, header='infer', names=None, index_col=None, usecols=None,
squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True,
parse_dates=False, infer_datetime_format=False, keep_date_col=False,
date_parser=None, dayfirst=False, cache_dates=True, iterator=False,
chunksize=None, compression='infer', thousands=None, decimal='.',
lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, dialect=None,
error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None)
    obj = MLConcepts()
    post_dict = obj.startPOSTProcess(df)
    get_dict = obj.startGETProcess(df)
    option_dict = obj.startOPTIONProcess(df)
    return
render(request,"users/UserMachineLearning.html",{'post_dict':post_dict,'get_d
ict':get_dict,"option_dict":option_dict})
```

userMachineLearningAlgorithm.py

```
from sklearn.model_selection import train_test_split
class MLConcepts:
    def startPOSTProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools',
'numOfIds','numOfBlobs','reqLen','isPOST']]
        #df_get = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs',
'reqLen', 'isGET']]
        X = df[['numOfParams', 'numOfBools',
'numOfIds','numOfBlobs','reqLen']]
        y = df[['isPOST']]
        X_train,X_test, y_train, y_test = train_test_split(X, y,
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('POST Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("POST Precession=", precision)
        from sklearn.metrics import recall_score
        recall = recall_score(y_pred.round(), y_test)
        print("POST Recall=", recall)
        post_dict = 
{"post_accuracy":accuracy,"post_precision":precision,"post_recall":recall}
        return post_dict

    def startGETProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools', 
'numOfIds','numOfBlobs','reqLen','isGET']]
        X = df[['numOfParams', 'numOfBools', 
'numOfIds','numOfBlobs','reqLen']]
        y = df[['isGET']]
        X_train,X_test, y_train, y_test = train_test_split(X, y, 
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('GET Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("GET Precession=", precision)
        from sklearn.metrics import recall_score
        recall = recall_score(y_pred.round(), y_test)
        print("GET Recall=", recall)
        get_dict = {"get_accuracy": accuracy, "get_precision":precision, 
"get_recall": recall}
        return get_dict

    def startOPTIONProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools', 
'numOfIds','numOfBlobs','reqLen','isOPTIONS']]
        X = df[['numOfParams', 'numOfBools', 
'numOfIds','numOfBlobs','reqLen']]
        y = df[['isOPTIONS']]
        X_train,X_test, y_train, y_test = train_test_split(X, y, 
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('OPTION Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("OPTION Precession=", precision)
        from sklearn.metrics import recall_score
        recall = recall_score(y_pred.round(), y_test)
        print("OPTION Recall=", recall)
        ooption_dict = {"option_accuracy": accuracy, 
"option_precision":precision, "option_recall": recall}
        return ooption_dict
```

models.py

```
from django.db import models

# Create your models here.

# Create your models here.
class UserRegistrationModel(models.Model):
    name = models.CharField(max_length=100)
    loginid = models.CharField(unique=True, max_length=100)
    password = models.CharField(max_length=100)
    mobile = models.CharField(unique=True, max_length=100)
    email = models.CharField(unique=True, max_length=100)
    locality = models.CharField(max_length=100)
    address = models.CharField(max_length=1000)
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    status = models.CharField(max_length=100)

    def __str__(self):
        return self.loginid

    class Meta:
        db_table = 'Registrations'

class UserSearchUrlModel(models.Model):
    id = models.AutoField(primary_key=True)
    urlname = models.CharField(max_length=250)
    depthfecth = models.IntegerField()
    c_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.urlname
    class Meta:
        db_table = "UserSearchUrls"

class CSRFResponse(models.Model):
    id = models.AutoField(primary_key=True)
    regex = models.CharField(max_length=10000)
    matches = models.CharField(max_length=100000)
    urlname = models.CharField(max_length=1000)
    c_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.urlname
    class Meta:
        db_table = "csrftables"
```

getting all csrfs:

```
from core.colors import green, yellow, end, run, good, info, bad, white, red

lightning = '\033[93;5m⚡\033[0m'


def banner():
    print ('''
     %s⚡ %sBOLT%s  ⚡%s
    ''' % (yellow, white, yellow, end))


banner()

try:
    import concurrent.futures
    try:
        from fuzzywuzzy import fuzz, process
    except:
        import os
        print ('%s fuzzywuzzy library is not installed, installing now.' %
info)
        os.system('pip3 install fuzzywuzzy')
        print ('%s fuzzywuzzy has been installed, please restart Bolt.' %
info)
        quit()
except:
    print ('%s Bolt is not compatible with python 2. Please run it with
python 3.' % bad)

import argparse
import json
import random
import re
import statistics

from core.entropy import isRandom
from core.datanize import datanize
from core.prompt import prompt
from core.photon import photon
from core.tweaker import tweaker
from core.evaluate import evaluate
from core.ranger import ranger
from core.zetanize import zetanize
from core.requester import requester
from core.utils import extractHeaders, strength, isProtected, stringToBinary,
longestCommonSubstring

parser = argparse.ArgumentParser()
parser.add_argument('-u', help='target url', dest='target')
parser.add_argument('-t', help='number of threads', dest='threads', type=int)
parser.add_argument('-l', help='levels to crawl', dest='level', type=int)
parser.add_argument('--delay', help='delay between requests',
                    dest='delay', type=int)
parser.add_argument('--timeout', help='http request timeout',
                    dest='timeout', type=int)
parser.add_argument('--headers', help='http headers',
                    dest='add_headers', nargs='?', const=True)
args = parser.parse_args()

if not args.target:
    print('\n' + parser.format_help().lower())
    quit()

if type(args.add_headers) == bool:
    headers = extractHeaders(prompt())
elif type(args.add_headers) == str:
    headers = extractHeaders(args.add_headers)
```

```
else:
    from core.config import headers

target = args.target
delay = args.delay or 0
level = args.level or 2
timeout = args.timeout or 20
threadCount = args.threads or 2

allTokens = []
weakTokens = []
tokenDatabase = []
insecureForms = []

print (' %s Phase: Crawling %s[%s1/6%s]%s' %
       (lightning, green, end, green, end))
dataset = photon(target, headers, level, threadCount)
allForms = dataset[0]
print ('\r%s Crawled %i URL(s) and found %i form(s).%-10s' %
       (info, dataset[1], len(allForms), ' '))
print (' %s Phase: Evaluating %s[%s2/6%s]%s' %
       (lightning, green, end, green, end))

evaluate(allForms, weakTokens, tokenDatabase, allTokens, insecureForms)

if weakTokens:
    print ('%s Weak token(s) found' % good)
    for weakToken in weakTokens:
        url = list(weakToken.keys())[0]
        token = list(weakToken.values())[0]
        print ('%s %s %s' % (info, url, token))

if insecureForms:
    print ('%s Insecure form(s) found' % good)
    for insecureForm in insecureForms:
        url = list(insecureForm.keys())[0]
        action = list(insecureForm.values())[0]['action']
        form = action.replace(target, '')
        if form:
            print ('%s %s %s[%s%s%s]%s' %
                   (bad, url, green, end, form, green, end))

print (' %s Phase: Comparing %s[%s3/6%s]%s' %
       (lightning, green, end, green, end))
uniqueTokens = set(allTokens)
if len(uniqueTokens) < len(allTokens):
    print ('%s Potential Replay Attack condition found' % good)
    print ('%s Verifying and looking for the cause' % run)
    replay = False
    for url, token in tokenDatabase:
        for url2, token2 in tokenDatabase:
            if token == token2 and url != url2:
                print ('%s The same token was used on %s%s%s and %s%s%s' %
                       (good, green, url, end, green, url2, end))
                replay = True
    if not replay:
        print ('%s Further investigation shows that it was a false
positive.')
```

```python
with open('./db/hashes.json') as f:
    hashPatterns = json.load(f)

if not allTokens:
    print ('%s No CSRF protection to test' % bad)
    quit()

aToken = allTokens[0]
matches = []
for element in hashPatterns:
    pattern = element['regex']
    if re.match(pattern, aToken):
        for name in element['matches']:
            matches.append(name)
if matches:
    print ('%s Token matches the pattern of following hash type(s):' % info)
    for name in matches:
        print ('    %s>%s %s' % (yellow, end, name))


def fuzzy(tokens):
    averages = []
    for token in tokens:
        sameTokenRemoved = False
        result = process.extract(token, tokens, scorer=fuzz.partial_ratio)
        scores = []
        for each in result:
            score = each[1]
            if score == 100 and not sameTokenRemoved:
                sameTokenRemoved = True
                continue
            scores.append(score)
        average = statistics.mean(scores)
        averages.append(average)
    return statistics.mean(averages)


try:
    similarity = fuzzy(allTokens)
    print ('%s Tokens are %s%i%%%s similar to each other on an average' %
           (info, green, similarity, end))
except statistics.StatisticsError:
    print ('%s No CSRF protection to test' % bad)
    quit()


def staticParts(allTokens):
    strings = list(set(allTokens.copy()))
    commonSubstrings = {}
    for theString in strings:
        strings.remove(theString)
        for string in strings:
            commonSubstring = longestCommonSubstring(theString, string)
            if commonSubstring not in commonSubstrings:
                commonSubstrings[commonSubstring] = []
            if len(commonSubstring) > 2:
                if theString not in commonSubstrings[commonSubstring]:
                    commonSubstrings[commonSubstring].append(theString)
                if string not in commonSubstrings[commonSubstring]:
                    commonSubstrings[commonSubstring].append(string)
    return commonSubstrings


result = {k: v for k, v in staticParts(allTokens).items() if v}

if result:
    print ('%s Common substring found' % info)
    print (json.dumps(result, indent=4))

simTokens = []

print (' %s Phase: Observing %s[%s4/6%s]%s' %
       (lightning, green, end, green, end))
print ('%s 100 simultaneous requests are being made, please wait.' % info)


def extractForms(url):
    response = requester(url, {}, headers, True, 0).text
    forms = zetanize(url, response)
    for each in forms.values():
        localTokens = set()
        inputs = each['inputs']
        for inp in inputs:
            value = inp['value']
            if value and match(r'^[\w\-_]+$', value):
                if strength(value) > 10:
                    simTokens.append(value)


while True:
    sample = random.choice(tokenDatabase)
    goodToken = list(sample.values())[0]
    if len(goodToken) > 0:
        goodCandidate = list(sample.keys())[0]
        break

threadpool = concurrent.futures.ThreadPoolExecutor(max_workers=30)
futures = (threadpool.submit(extractForms, goodCandidate)
           for goodCandidate in [goodCandidate] * 30)
for i in concurrent.futures.as_completed(futures):
    pass

if simTokens:
    if len(set(simTokens)) < len(simTokens):
        print ('%s Same tokens were issued for simultaneous requests.' %
good)
    else:
        print (simTokens)
else:
    print ('%s Different tokens were issued for simultaneous requests.' %
info)

print (' %s Phase: Testing %s[%s5/6%s]%s' %
       (lightning, green, end, green, end))

parsed = ''
print ('%s Finding a suitable form for further testing. It may take a while.'
% run)
```

```
for url, forms in allForms[0].items():
    found = False
    parsed = datanize(forms, tolerate=True)
    if parsed:
        found = True
        break
    if found:
        break

if not parsed:
    candidate = list(random.choice(tokenDatabase).keys())[0]
    parsed = datanize(candidate, headers, tolerate=True)
    print (parsed)

origGET = parsed[0]
origUrl = parsed[1]
origData = parsed[2]

print ('%s Making a request with CSRF token for comparison.' % run)
response = requester(origUrl, origData, headers, origGET, 0)
originalCode = response.status_code
originalLength = len(response.text)
print ('%s Status Code: %s' % (info, originalCode))
print ('%s Content Length: %i' % (info, originalLength))
print ('%s Checking if the resonse is dynamic.' % run)
response = requester(origUrl, origData, headers, origGET, 0)
secondLength = len(response.text)
if originalLength != secondLength:
    print ('%s Response is dynamic.' % info)
    tolerableDifference = abs(originalLength - secondLength)
else:
    print ('%s Response isn\'t dynamic.' % info)
    tolerableDifference = 0

print ('%s Emulating a mobile browser' % run)
print ('%s Making a request with mobile browser' % run)
headers['User-Agent'] = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows CE; PPC;
240x320)'
response = requester(origUrl, {}, headers, True, 0).text
parsed = zetanize(origUrl, response)
if isProtected(parsed):
    print ('%s CSRF protection is enabled for mobile browsers as well.' %
bad)
else:
    print ('%s CSRF protection isn\'t enabled for mobile browsers.' % good)

print ('%s Making a request without CSRF token parameter.' % run)

data = tweaker(origData, 'remove')
response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)
```

```python
print ('%s Making a request without CSRF token parameter value.' % run)
data = tweaker(origData, 'clear')

response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)


seeds = ranger(allTokens)

print ('%s Checking if tokens are checked to a specific length' % run)

for index in range(len(allTokens[0])):
    data = tweaker(origData, 'replace', index=index, seeds=seeds)
    response = requester(origUrl, data, headers, origGET, 0)
    if response.status_code == originalCode:
        if str(originalCode)[0] in ['4', '5']:
            break
        else:
            difference = abs(originalLength - len(response.text))
            if difference <= tolerableDifference:
                print ('%s Last %i chars of token aren\'t being checked' %
                       (good, index + 1))
    else:
        break

print ('%s Generating a fake token.' % run)

data = tweaker(origData, 'generate', seeds=seeds)
print ('%s Making a request with the self generated token.' % run)

response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)

print (' %s Phase: Analysing %s[%s6/6%s]%s' %
       (lightning, green, end, green, end))

binary = stringToBinary(''.join(allTokens))
result = isRandom(binary)
for name, result in result.items():
    if not result:
        print ('%s %s : %s%s%s' % (good, name, green, 'non-random', end))
    else:
        print ('%s %s : %s%s%s' % (bad, name, red, 'random', end))
```

Admin side Views.py

```
from django.shortcuts import render,HttpResponse
from django.contrib import messages
from users.models import UserRegistrationModel,CSRFResponse
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
import pandas as pd
# Create your views here.
def AdminLoginCheck(request):
    if request.method == 'POST':
        usrid = request.POST.get('loginname')
        pswd = request.POST.get('pswd')
        print("User ID is = ", usrid)
        if usrid == 'admin' and pswd == 'admin':
            return render(request, 'admins/AdminHome.html')
        else:
            messages.success(request, 'Please Check Your Login Details')
    return render(request, 'AdminLogin.html', {})

def AdminHome(request):
    return render(request, 'admins/AdminHome.html')

def AdminViewUsers(request):
    data = UserRegistrationModel.objects.all()
    return render(request, 'admins/RegisteredUsers.html', {'data': data})

def AdminActivaUsers(request):
    if request.method == 'GET':
        id = request.GET.get('uid')
        status = 'activated'
        print("PID = ", id, status)
        UserRegistrationModel.objects.filter(id=id).update(status=status)
        data = UserRegistrationModel.objects.all()
        return render(request,'admins/RegisteredUsers.html',{'data':data})


def adminviewallcsrfs(request):
    data_list = CSRFResponse.objects.all()
    page = request.GET.get('page', 1)

    paginator = Paginator(data_list, 60)
    try:
        users = paginator.page(page)
    except PageNotAnInteger:
        users = paginator.page(1)
    except EmptyPage:
        users = paginator.page(paginator.num_pages)

    return render(request, 'admins/viewAllCSRFS.html', {'users': users})

def PostRequestdata(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',',
delimiter=None, header='infer', names=None,
                     index_col=None, usecols=None, squeeze=False,
prefix=None, mangle_dupe_cols=True, dtype=None,
                     engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False,
                          skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True,
                          verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False,
                          keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False,
                          chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None,
                          quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None,
                          dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=True,
                          memory_map=False, float_precision=None)
    data = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs',
'reqLen', 'isPOST']]
    data = data.to_html()
    #print(data)

    return render(request, "admins/PostviewData.html",{"data":data})

def GetRequestdata(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',',
delimiter=None, header='infer', names=None,
                          index_col=None, usecols=None, squeeze=False,
prefix=None, mangle_dupe_cols=True, dtype=None,
                          engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False,
                          skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True,
                          verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False,
                          keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False,
                          chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None,
                          quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None,
                          dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=True,
                          memory_map=False, float_precision=None)
    data = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs',
'reqLen', 'isGET']]
    data = data.to_html()

    return render(request, "admins/GetviewData.html", {"data": data})
```

All Ursl.py

```
"""WebVulnerability URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
```

```
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from WebVulnerability import views as mainView
from users import views as usr
from admins import views as admins
from django.contrib.staticfiles.urls import static
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path("index/", mainView.logout, name="index"),
    path("", mainView.index, name="index"),
    path("logout/", mainView.logout, name="logout"),
    path("UserLogin/", mainView.UserLogin, name="UserLogin"),
    path("AdminLogin/", mainView.AdminLogin, name="AdminLogin"),
    path("UserRegister/", mainView.UserRegister, name="UserRegister"),

    #### User Views #####
    path("UserRegisterActions/", usr.UserRegisterActions,
name="UserRegisterActions"),
    path("UserLoginCheck/", usr.UserLoginCheck, name="UserLoginCheck"),
    path("UserHome/",usr.UserHome, name="UserHome"),
    path("UserPreProcessForm/", usr.UserPreProcessForm,
name="UserPreProcessForm"),
    path("UserCSRFProcessByBolt/", usr.UserCSRFProcessByBolt,
name="UserCSRFProcessByBolt"),
    path("UserMitchProcess/", usr.UserMitchProcess, name="UserMitchProcess"),
    path("UserMachineLearning/", usr.UserMachineLearning,
name="UserMachineLearning"),



    #####Admin Side Views #######
    path("AdminLoginCheck/", admins.AdminLoginCheck, name="AdminLoginCheck"),
    path("AdminHome/", admins.AdminHome, name="AdminHome"),
    path("AdminViewUsers/", admins.AdminViewUsers, name="AdminViewUsers"),
    path("AdminActivaUsers/", admins.AdminActivaUsers,
name="AdminActivaUsers"),
    path("adminviewallcsrfs/", admins.adminviewallcsrfs,
name="adminviewallcsrfs"),
    path("PostRequestdata/", admins.PostRequestdata, name="PostRequestdata"),
    path("GetRequestdata/", admins.GetRequestdata, name="GetRequestdata"),




]
urlpatterns += staticfiles_urlpatterns()
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Base.html

```
{%load static%}
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, 
shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="TemplateMo">
    <link 
href="https://fonts.googleapis.com/css?family=Poppins:100,200,300,400,500,600
,700,800,900&display=swap" rel="stylesheet">
    <title>Finance Business HTML5 Template</title>
    <!-- Bootstrap core CSS -->
    <link href="{%static 'vendor/bootstrap/css/bootstrap.min.css'%}" 
rel="stylesheet">
    <!-- Additional CSS Files -->
    <link rel="stylesheet" href="{%static 'assets/css/fontawesome.css'%}">
    <link rel="stylesheet" href="{%static 'assets/css/templatemo-finance-
business.css'%}">
    <link rel="stylesheet" href="{%static 'assets/css/owl.css'%}">
  </head>

  <body>
    <div id="preloader">
        <div class="jumper">
            <div></div>
            <div></div>
            <div></div>
        </div>
    </div>
    <header class="">
      <nav class="navbar navbar-expand-lg">
        <div class="container">
          <a class="navbar-brand" href="index.html"><h2>Web 
Vulnerability</h2></a>
          <button class="navbar-toggler" type="button" data-toggle="collapse" 
data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-
expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
              <li class="nav-item">
                <a class="nav-link" href="{%url 'index'%}">Home</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{%url 'UserLogin'%}">User</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{%url 'AdminLogin'%}">Admin</a>
              </li>

              <li class="nav-item">
                <a class="nav-link" href="{%url 
'UserRegister'%}">Registrations</a>
              </li>
            </ul>
          </div>
        </div>
      </nav>
    </header>
    <!-- Page Content -->
    <!-- Banner Starts Here -->
    {%block contents%}

    {%endblock%}


    <div class="services">
      <div class="container">
        <div class="row">
          <div class="col-md-12">
            <div class="section-heading">
              <h2>Web Vulnerability  <em>Detection</em></h2>
              <span>use supervised learning to automatically train a
classifier
which partitions selected web objects of interest, e.g.,
HTTP requests, HTTP responses or cookies, based on
the web application semantics. For example, in the case
of CSRF detection, the classifier would be used to
identify security-sensitive HTTP requests</span>
            </div>
          </div>
                    </div>
      </div>
    </div>
    <div class="sub-footer">
      <div class="container">
        <div class="row">
          <div class="col-md-12">
            <p>Copyright &copy; 2020 Alex and Co., Ltd.

            - Design: <a rel="nofollow noopener" href="Google"
target="_blank">TemplateMo</a></p>
          </div>
        </div>
      </div>
    </div>

    <!-- Bootstrap core JavaScript -->
    <script src="{% static 'vendor/jquery/jquery.min.js'%}"></script>
    <script src="{% static
'vendor/bootstrap/js/bootstrap.bundle.min.js'%}"></script>

    <!-- Additional Scripts -->
    <script src="{% static 'assets/js/custom.js'%}"></script>
    <script src="{% static 'assets/js/owl.js'%}"></script>
    <script src="{% static 'assets/js/slick.js'%}"></script>
    <script src="{% static 'assets/js/accordions.js'%}"></script>

    <script language = "text/Javascript">
      cleared[0] = cleared[1] = cleared[2] = 0; //set a cleared flag for each
field
      function clearField(t){                   //declaring the array outside
of the
      if(! cleared[t.id]){                      // function makes it static
and global
          cleared[t.id] = 1;  // you could use true and false, but that's
more typing
          t.value='';         // with more chance of typos
```

```
            t.style.color='#fff';
            }
        }
    </script>

  </body>
</html>
```

UserRegistrations.html

```
{%extends 'base.html'%}

{%block contents%}

<div class="main-banner header-text" id="top">
        <div class="Modern-Slider">
          <div class="item item-1">
            <div class="img-fill">
                <center>
                <div class="text-content"><br/><br/><br/><br/>
                  <h4>User Registration Here</h4>
                  <p>
                      <form action="{%url 'UserRegisterActions'%}" 
method="POST" class="text-primary" style="width:100%">
                                            {% csrf_token %}
                                            <table>
                                                <tr>
                                                    <td class="text-primary">Customer 
Name</td>

                                                    <td>{{form.name}}</td>
                                                </tr>
                                                <tr>
                                                    <td>Login ID</td>
                                                    <td>{{form.loginid}}</td>
                                                </tr>
                                                <tr>
                                                    <td>Password</td>
                                                    <td>{{form.password}}</td>
                                                </tr>
                                                <tr>
                                                    <td>Mobile</td>
                                                    <td>{{form.mobile}}</td>
                                                </tr>
                                                <tr>
                                                    <td>email</td>
                                                    <td>{{form.email}}</td>
                                                </tr>
                                                <tr>
                                                    <td>Locality</td>
                                                    <td>{{form.locality}}</td>
                                                </tr>
                                                <tr>
                                                    <td>Address</td>
                                                    <td>{{form.address}}</td>
                                                </tr>
                                                <tr>
                                                    <td>City</td>
                                                    <td>{{form.city}}</td>
                                                </tr>
```

```
                                        <tr>
                                            <td>State</td>
                                            <td>{{form.state}}</td>
                                        </tr>
                                        <tr>
                                            <td></td>
                                            <td>{{form.status}}</td>
                                        </tr>

                                        <tr>
                                            <td>
                                                <button class="btn btn-
primary my-2 my-sm-0" style="margin-left:20%;"
                                                        type="submit">
                                                    Register
                                                </button>
                                            </td>
                                        </tr>

                                        {% if messages %}
                                        {% for message in messages %}
                                        <font color='GREEN'> {{ message
}}</font>
                                        {% endfor %}
                                        {% endif %}

                                    </table>

                                </form>
                  </p>
                </div>
                    </center>
            </div>
          </div>
        </div>
    </div>


{%endblock%}
```

CHAPTER - 7
TESTING

7.1 Testing

Methodologies Test

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

Types of Tests

- **Unit Testing**

  Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

- **Integration Testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components. The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

- **Functional Test**

Functional Test provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation and user manuals.

Functional testing is centered on the following items:

- Valid Input - Identified classes of valid input must be accepted.
- Invalid Input - Identified classes of invalid input must be rejected.
- Functions - Identified functions must be exercised.
- Output - Identified classes of application outputs must be exercised.
- Systems/Procedures - Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

- **System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

- **White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

- **Black Box Testing**

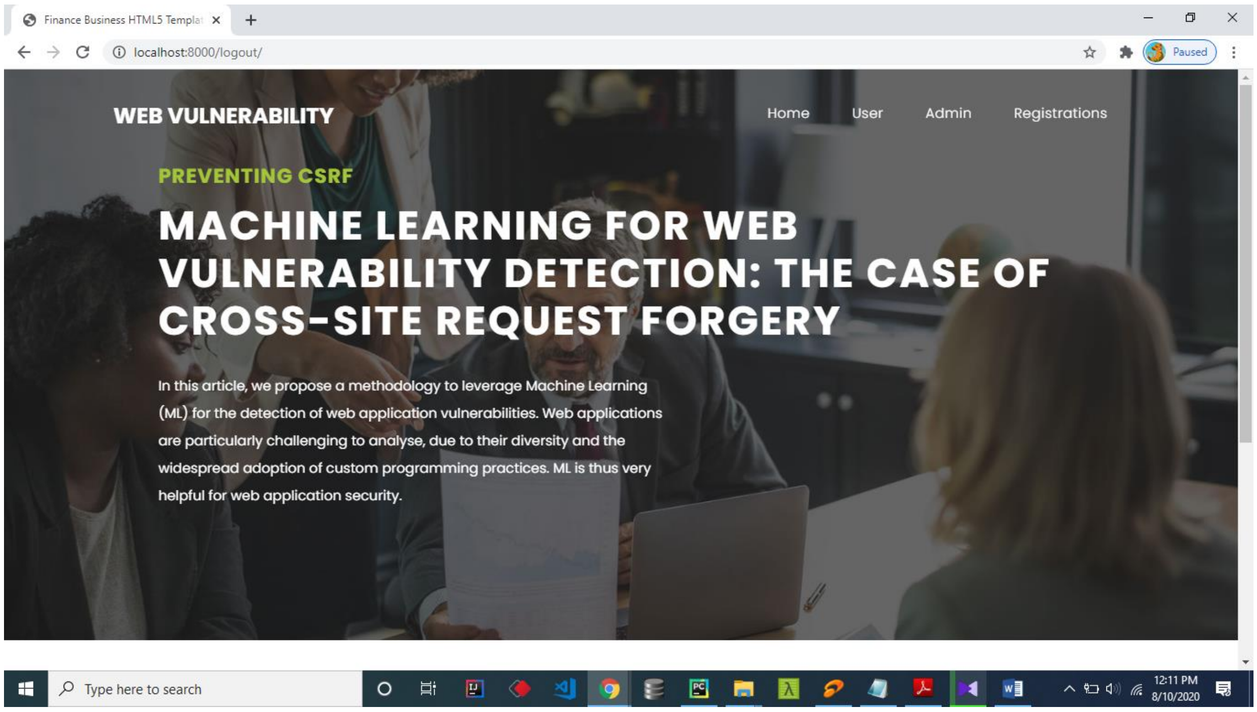
Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box, you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

S.no	Test Case	Excepted Result	Result	Remarks(IF Fails)
1.	User Register	If User registration successfully.	Pass	If already user email exist then it fails.
2.	User Login	If User name and password is correct then it will getting valid page.	Pass	Un Register Users will not logged in.
3.	Test URLS	User has to give the a valid urls which is running over the internet then it will fetch	Pass	If the url is not available then we will get errors
4.	Store all info into json file	The csrf token and its format are stored in json file	Pass	File location must be available at db folder other wise we will get errors
5.	Load the Mitch Dataset	Mitch Dataset loaded	Pass	In the current path if dataset not available then we will get errors.
6.	Test with Post Method	Loaded dataset test with post method then only accuracy will be calculated	Pass	Data classified table required
7.	Test with GET Method	Loaded dataset test with post method then only accuracy will be calculated	Pass	Data classified table required
8.	Predict Train and Test data	Predicted and original salary will be displayed	Pass	Trains and test size must be specify otherwise failed
9.	Admin login	Admin can login with his login credential. If success he get his home page	Pass	Invalid login details will not be allowed here
10.	Admin can activate the register users	Admin can activate the register user id	Pass	If user id not found then it won't login.

Table 7.1 Test Cases

CHAPTER – 8

OUTPUT SCREENS



MACHINE LEARNING FOR WEB VULNERABILITY DETECTION: THE CASE OF CROSS-SITE REQUEST FORGERY

In this article, we propose a methodology to leverage Machine Learning (ML) for the detection of web application vulnerabilities. Web applications are particularly challenging to analyse, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful for web application security.

Fig – 8.1 Home page

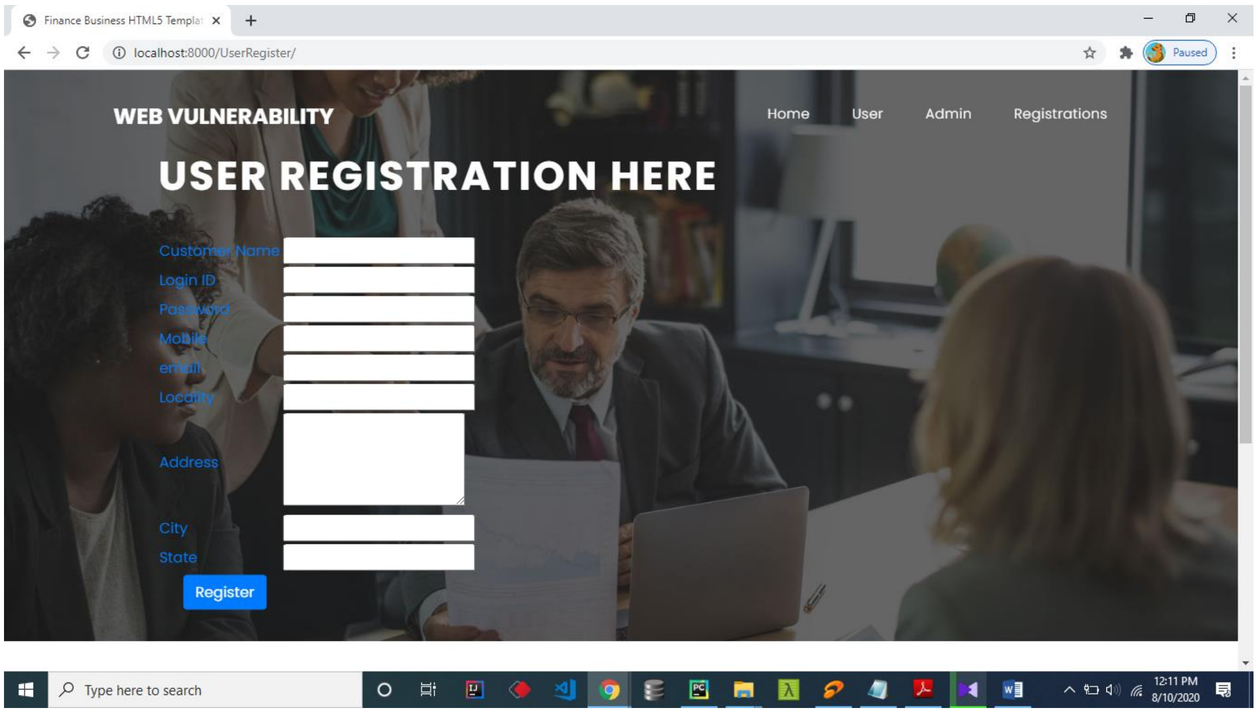


Fig – 8.2 User Registration page

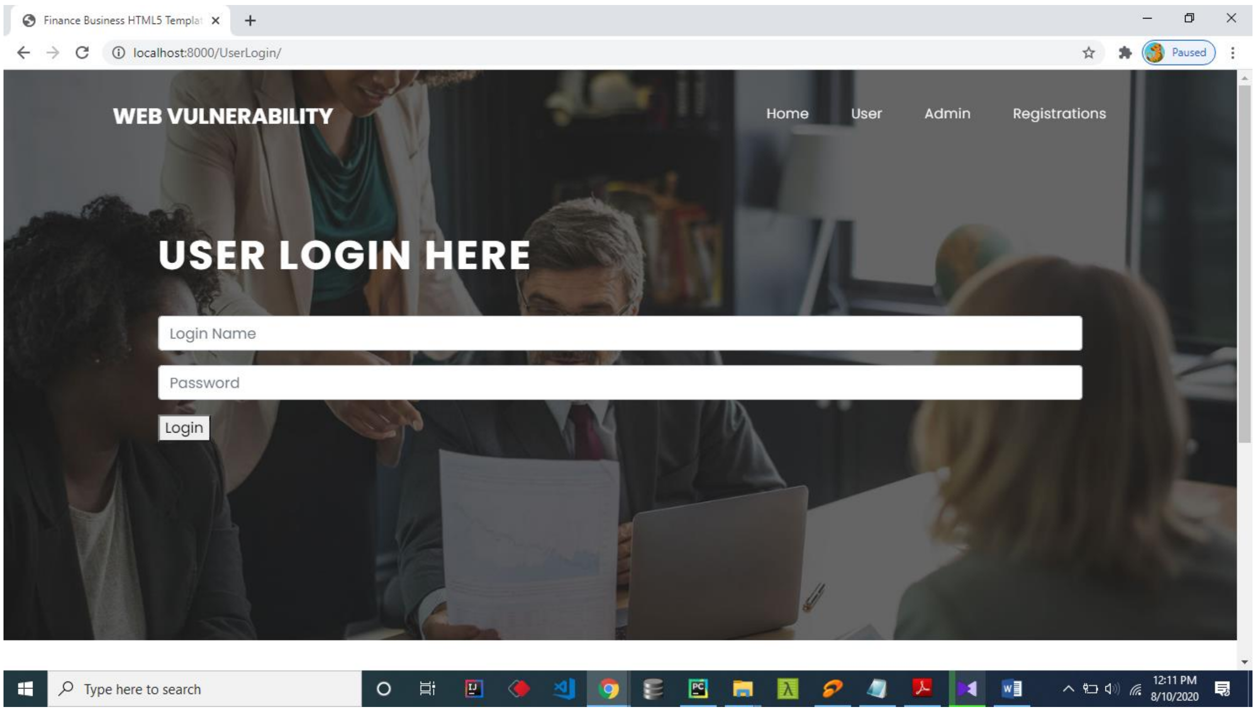
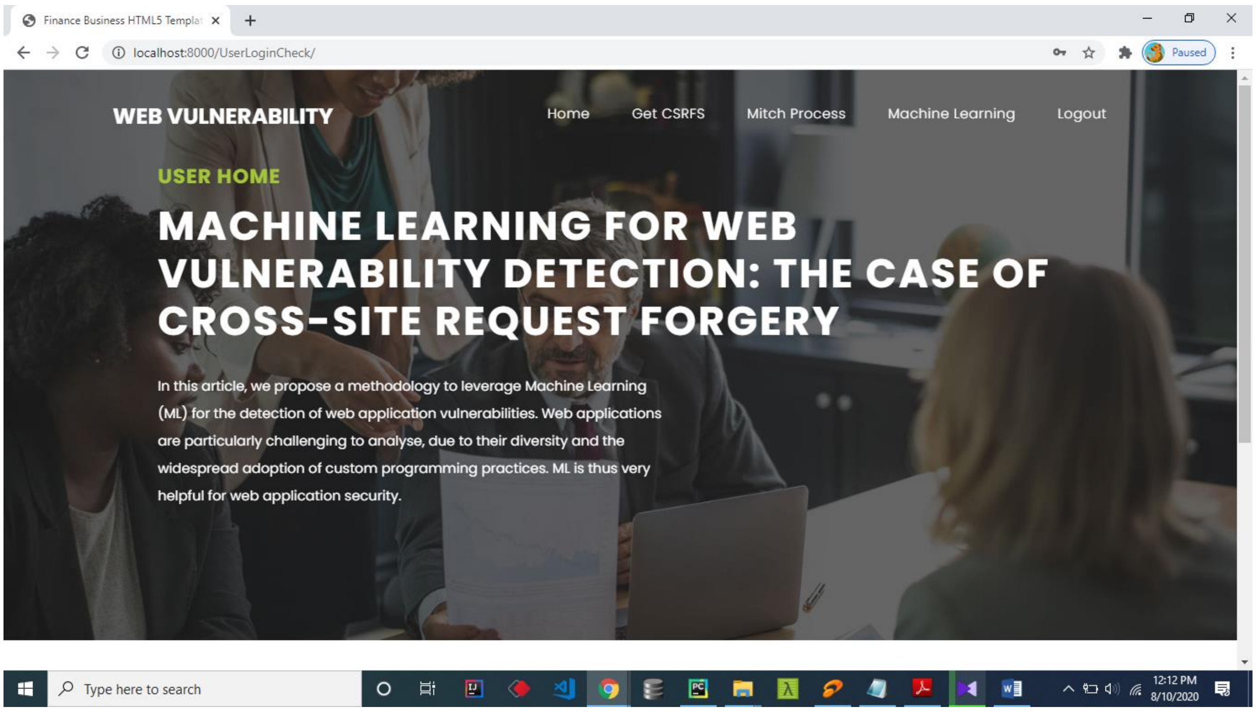


Fig – 8.3 User Login page



MACHINE LEARNING FOR WEB VULNERABILITY DETECTION: THE CASE OF CROSS-SITE REQUEST FORGERY

In this article, we propose a methodology to leverage Machine Learning (ML) for the detection of web application vulnerabilities. Web applications are particularly challenging to analyse, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful for web application security.

Fig – 8.4 User Home Page

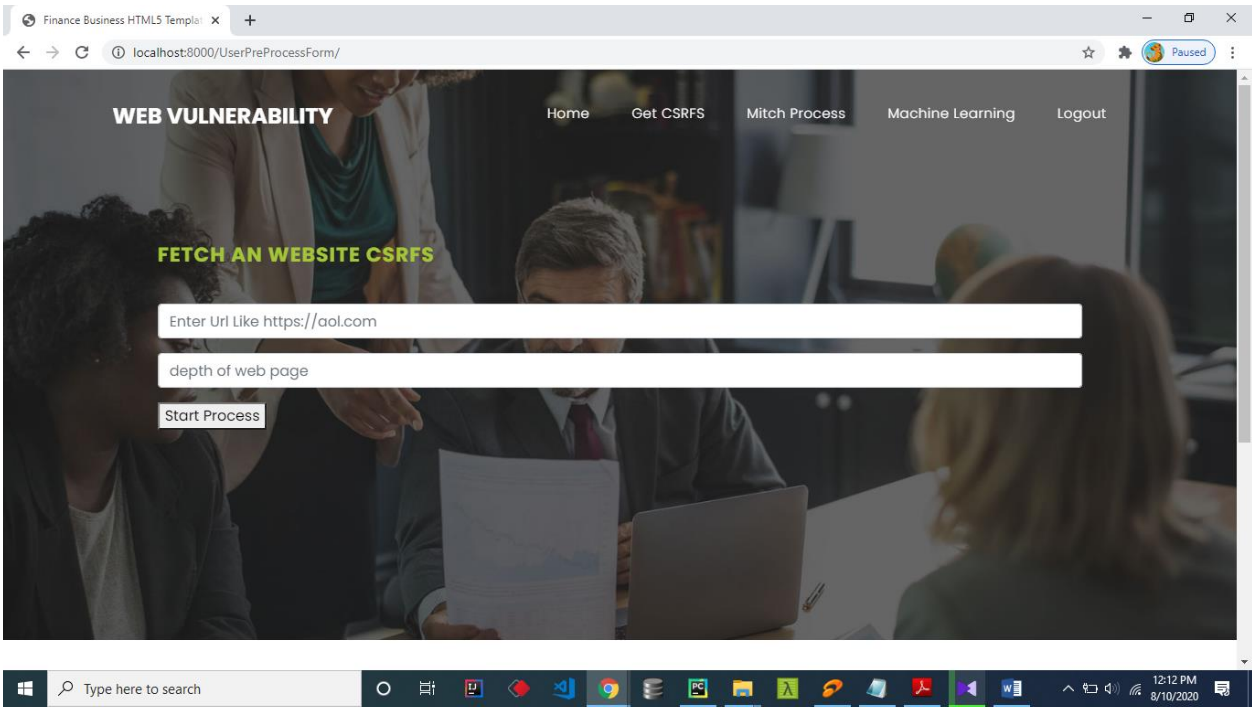
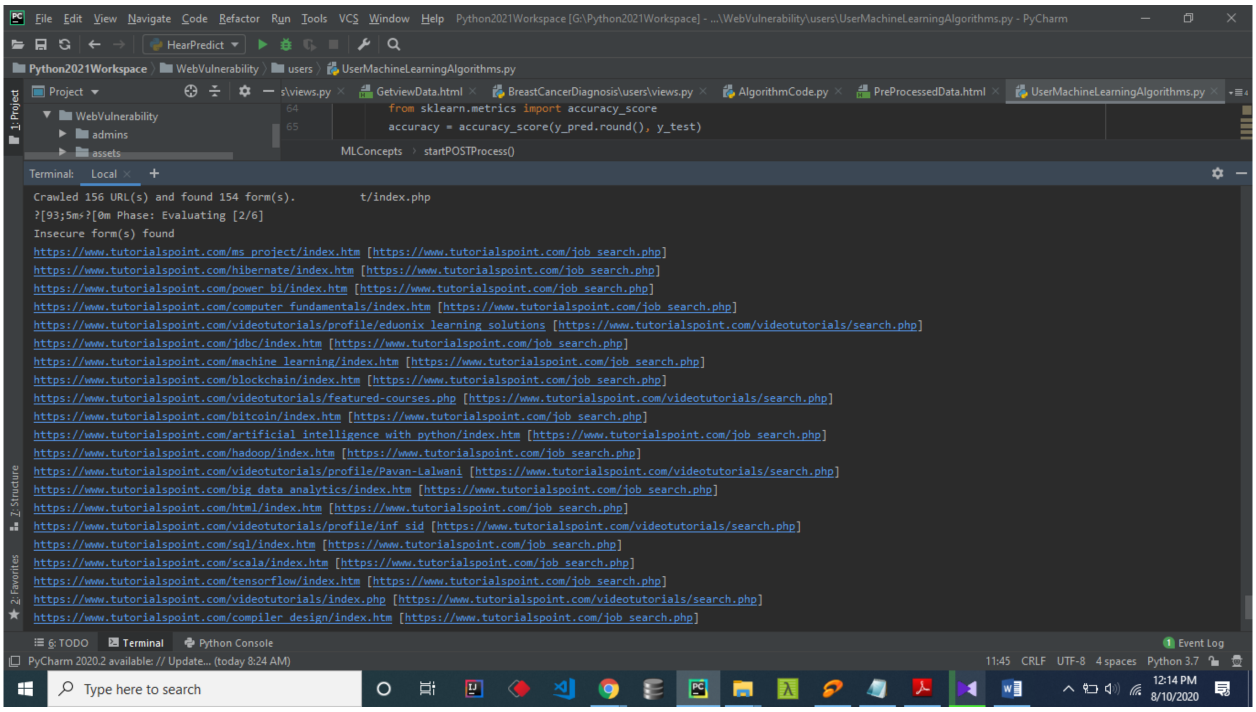
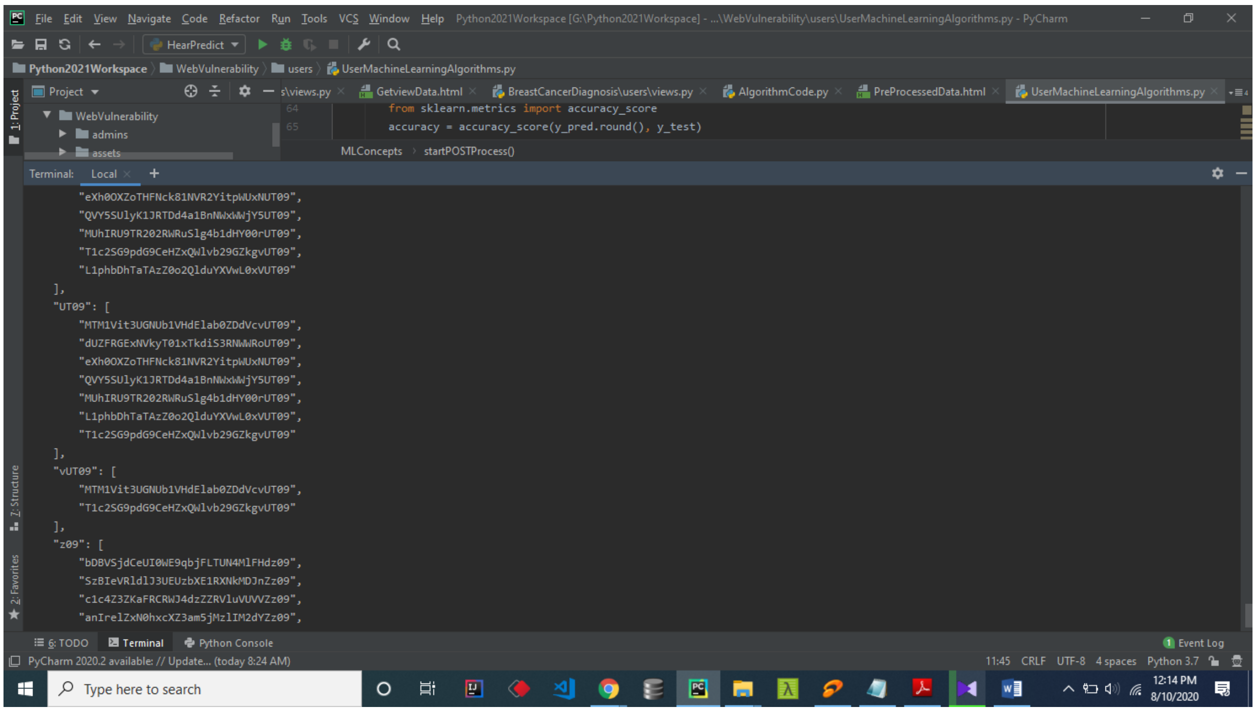


Fig – 8.5 Getting website csrfs


```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_pred.round(), y_test)
```

```
Crawled 156 URL(s) and found 154 form(s).         t/index.php
?[93;5m⚡?[0m Phase: Evaluating [2/6]
Insecure form(s) found
https://www.tutorialspoint.com/ms_project/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/hibernate/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/power_bi/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/computer_fundamentals/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/videotutorials/profile/eduonix_learning_solutions [https://www.tutorialspoint.com/videotutorials/search.php]
https://www.tutorialspoint.com/jdbc/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/machine_learning/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/blockchain/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/videotutorials/featured-courses.php [https://www.tutorialspoint.com/videotutorials/search.php]
https://www.tutorialspoint.com/bitcoin/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/artificial_intelligence_with_python/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/hadoop/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/videotutorials/profile/Pavan-Lalwani [https://www.tutorialspoint.com/videotutorials/search.php]
https://www.tutorialspoint.com/big_data_analytics/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/html/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/videotutorials/profile/inf_sid [https://www.tutorialspoint.com/videotutorials/search.php]
https://www.tutorialspoint.com/sql/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/scala/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/tensorflow/index.htm [https://www.tutorialspoint.com/job_search.php]
https://www.tutorialspoint.com/videotutorials/index.php [https://www.tutorialspoint.com/videotutorials/search.php]
https://www.tutorialspoint.com/compiler_design/index.htm [https://www.tutorialspoint.com/job_search.php]
```

Fig – 8.6 Scanning urls

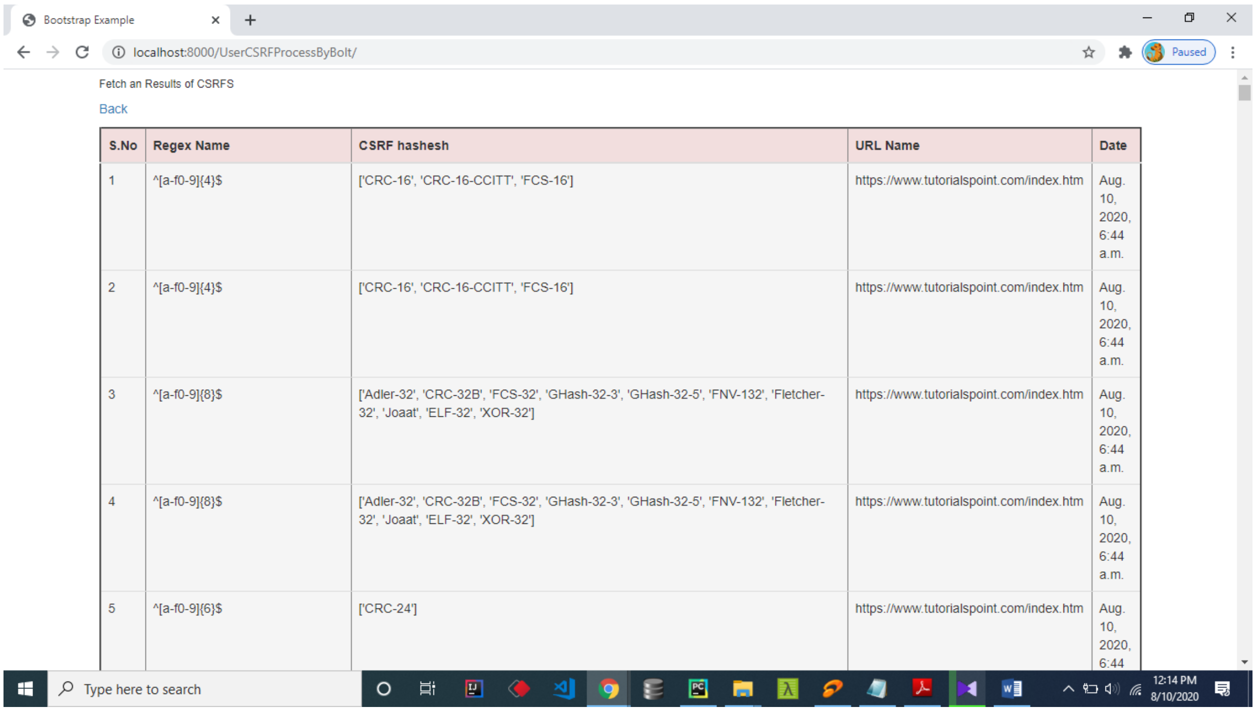


```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_pred.round(), y_test)
```

```
        "eXh0OXZoTHFNck81NVR2YitpWUxNUT09",
        "QVY5SUlyK1JRTDd4a1BnNWxWWjY5UT09",
        "MUhIRU9TR202RWRuSlg4b1dHY00rUT09",
        "T1c2SG9pdG9CeHZxQWlvb29GZkgvUT09",
        "L1phbDhTaTAzZ0o2QlduYXVwL0xVUT09"
    ],
    "UT09": [
        "MTM1Vit3UGNUb1VHdElab0ZDdVcvUT09",
        "dUZFRGExNVkyT01xTkdiS3RNWWRoUT09",
        "eXh0OXZoTHFNck81NVR2YitpWUxNUT09",
        "QVY5SUlyK1JRTDd4a1BnNWxWWjY5UT09",
        "MUhIRU9TR202RWRuSlg4b1dHY00rUT09",
        "L1phbDhTaTAzZ0o2QlduYXVwL0xVUT09",
        "T1c2SG9pdG9CeHZxQWlvb29GZkgvUT09"
    ],
    "vUT09": [
        "MTM1Vit3UGNUb1VHdElab0ZDdVcvUT09",
        "T1c2SG9pdG9CeHZxQWlvb29GZkgvUT09"
    ],
    "z09": [
        "bDBVSjdCeUI0WE9qbjFLTUN4MlFHdz09",
        "SzBIeVRldlJ3UEUzbXE1RXNkMDJnZz09",
        "c1c4Z3ZKaFRCRWJ4dzZZRVluVUVVZz09",
        "anIrelZxN0hxcXZ3am5jMzlIM2dYZz09",
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Python2021Workspace [G:\Python2021Workspace] - ...\WebVulnerability\users\UserMachineLearningAlgorithms.py - PyCharm
HearPredict
Python2021Workspace › WebVulnerability › users › UserMachineLearningAlgorithms.py
Project
WebVulnerability
admins
assets
s\views.py
GetviewData.html
BreastCancerDiagnosis\users\views.py
AlgorithmCode.py
PreProcessedData.html
UserMachineLearningAlgorithms.py
MLConcepts › startPOSTProcess()
Terminal: Local
6: TODO
Terminal
Python Console
Event Log
PyCharm 2020.2 available: // Update... (today 8:24 AM)
11:45 CRLF UTF-8 4 spaces Python 3.7
Type here to search
12:14 PM
8/10/2020

Fig – 8.7 CSRF token

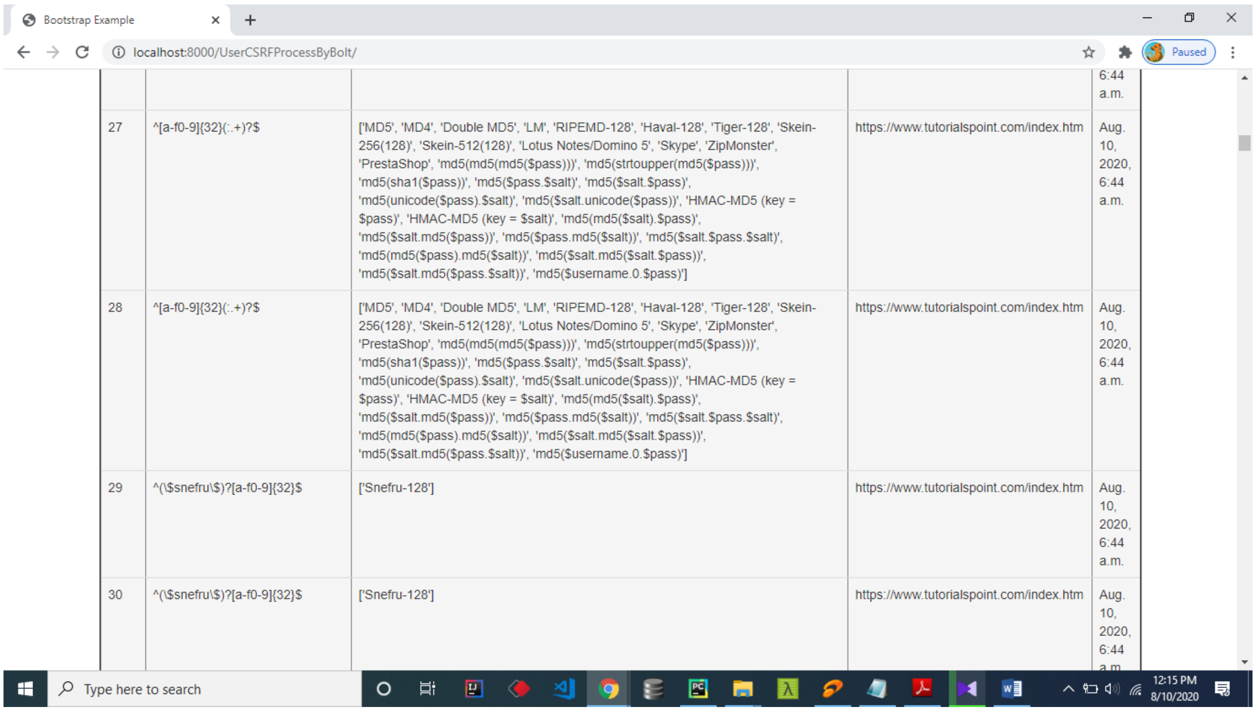


Fetch an Results of CSRFS

Back

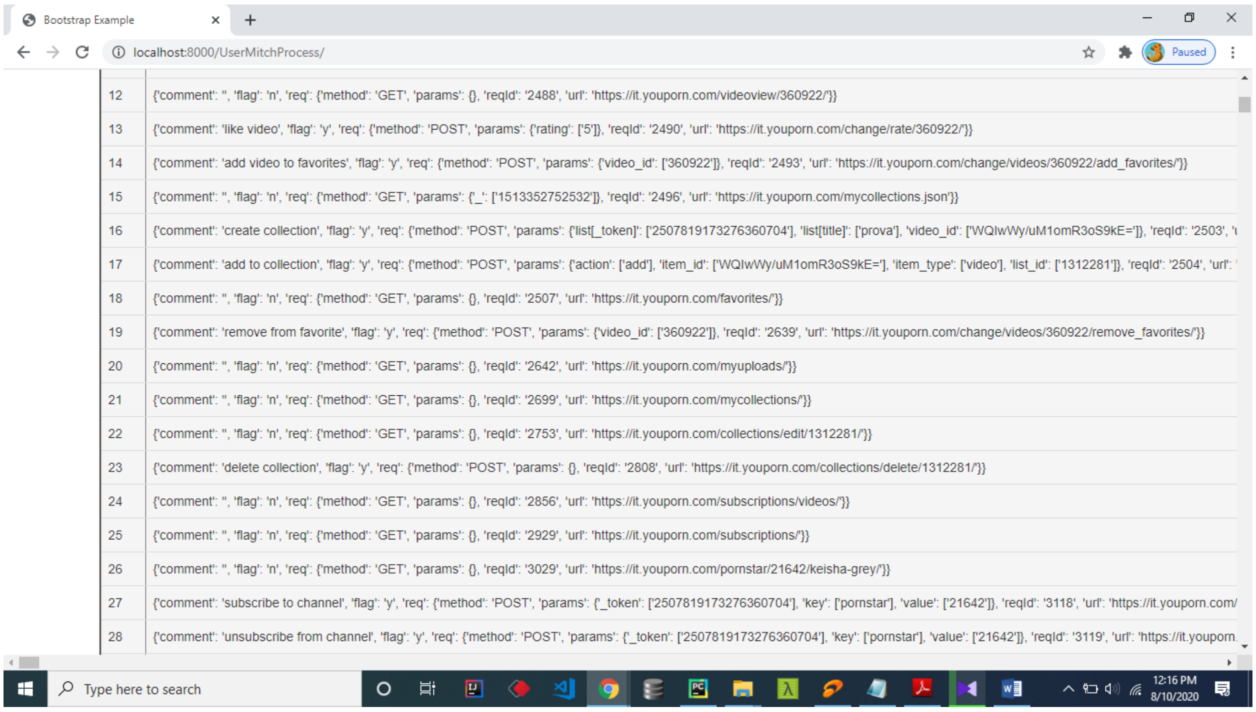
S.No	Regex Name	CSRF hashesh	URL Name	Date
1	^[a-f0-9]{4}$	['CRC-16', 'CRC-16-CCITT', 'FCS-16']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
2	^[a-f0-9]{4}$	['CRC-16', 'CRC-16-CCITT', 'FCS-16']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
3	^[a-f0-9]{8}$	['Adler-32', 'CRC-32B', 'FCS-32', 'GHash-32-3', 'GHash-32-5', 'FNV-132', 'Fletcher-32', 'Joaat', 'ELF-32', 'XOR-32']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
4	^[a-f0-9]{8}$	['Adler-32', 'CRC-32B', 'FCS-32', 'GHash-32-3', 'GHash-32-5', 'FNV-132', 'Fletcher-32', 'Joaat', 'ELF-32', 'XOR-32']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
5	^[a-f0-9]{6}$	['CRC-24']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44

Fig – 8.8 Given website csrf results



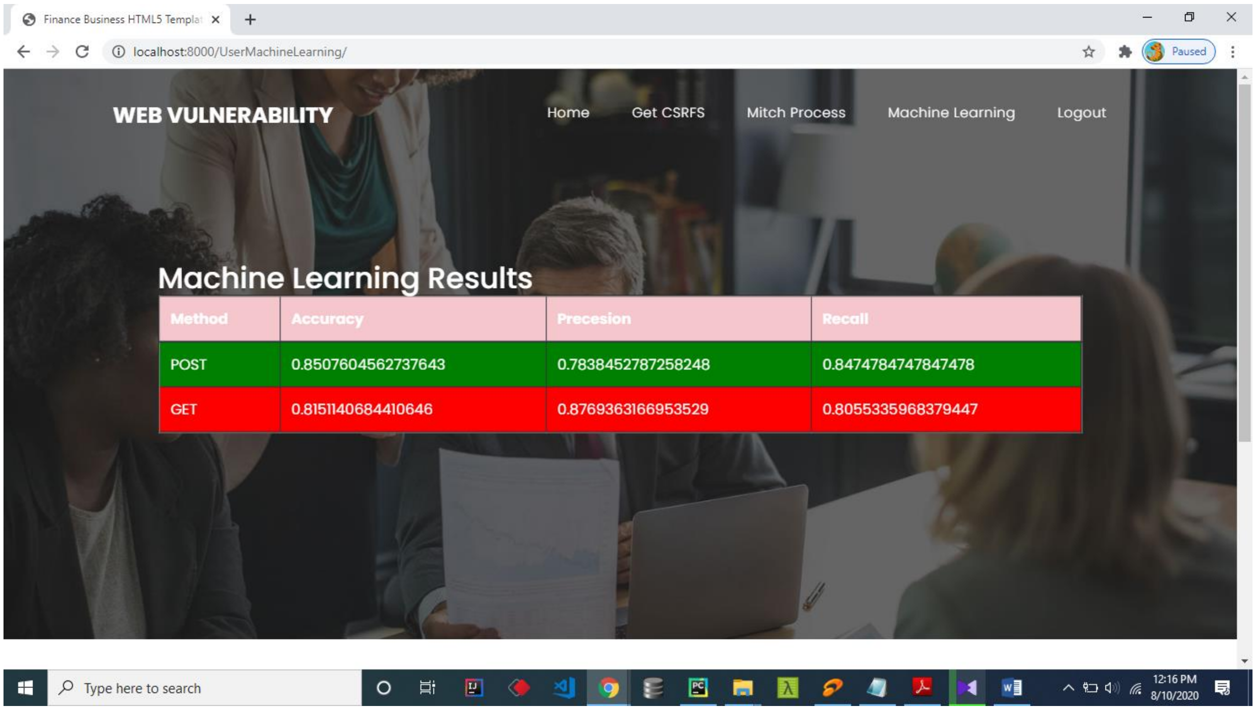
				6:44 a.m.
27	^[a-f0-9]{32}(:.+)?$	['MD5', 'MD4', 'Double MD5', 'LM', 'RIPEMD-128', 'Haval-128', 'Tiger-128', 'Skein-256(128)', 'Skein-512(128)', 'Lotus Notes/Domino 5', 'Skype', 'ZipMonster', 'PrestaShop', 'md5(md5(md5($pass)))', 'md5(strtoupper(md5($pass)))', 'md5(sha1($pass))', 'md5($pass.$salt)', 'md5($salt.$pass)', 'md5(unicode($pass).$salt)', 'md5($salt.unicode($pass))', 'HMAC-MD5 (key = $pass)', 'HMAC-MD5 (key = $salt)', 'md5(md5($salt).$pass)', 'md5($salt.md5($pass))', 'md5($pass.md5($salt))', 'md5($salt.$pass.$salt)', 'md5(md5($pass).md5($salt))', 'md5($salt.md5($salt.$pass))', 'md5($salt.md5($pass.$salt))', 'md5($username.0.$pass)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
28	^[a-f0-9]{32}(:.+)?$	['MD5', 'MD4', 'Double MD5', 'LM', 'RIPEMD-128', 'Haval-128', 'Tiger-128', 'Skein-256(128)', 'Skein-512(128)', 'Lotus Notes/Domino 5', 'Skype', 'ZipMonster', 'PrestaShop', 'md5(md5(md5($pass)))', 'md5(strtoupper(md5($pass)))', 'md5(sha1($pass))', 'md5($pass.$salt)', 'md5($salt.$pass)', 'md5(unicode($pass).$salt)', 'md5($salt.unicode($pass))', 'HMAC-MD5 (key = $pass)', 'HMAC-MD5 (key = $salt)', 'md5(md5($salt).$pass)', 'md5($salt.md5($pass))', 'md5($pass.md5($salt))', 'md5($salt.$pass.$salt)', 'md5(md5($pass).md5($salt))', 'md5($salt.md5($salt.$pass))', 'md5($salt.md5($pass.$salt))', 'md5($username.0.$pass)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
29	^(\$snefru\$)?[a-f0-9]{32}$	['Snefru-128']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
30	^(\$snefru\$)?[a-f0-9]{32}$	['Snefru-128']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.

Fig – 8.9 MD5 Token



12	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2488', 'url': 'https://it.youporn.com/videoview/360922/'}}
13	{'comment': 'like video', 'flag': 'y', 'req': {'method': 'POST', 'params': {'rating': ['5']}, 'reqId': '2490', 'url': 'https://it.youporn.com/change/rate/360922/'}}
14	{'comment': 'add video to favorites', 'flag': 'y', 'req': {'method': 'POST', 'params': {'video_id': ['360922']}, 'reqId': '2493', 'url': 'https://it.youporn.com/change/videos/360922/add_favorites/'}}
15	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {'_': ['1513352752532']}, 'reqId': '2496', 'url': 'https://it.youporn.com/mycollections.json'}}
16	{'comment': 'create collection', 'flag': 'y', 'req': {'method': 'POST', 'params': {'list[_token]': ['2507819173276360704'], 'list[title]': ['prova'], 'video_id': ['WQIwWy/uM1omR3oS9kE=']}, 'reqId': '2503', '
17	{'comment': 'add to collection', 'flag': 'y', 'req': {'method': 'POST', 'params': {'action': ['add'], 'item_id': ['WQIwWy/uM1omR3oS9kE='], 'item_type': ['video'], 'list_id': ['1312281']}, 'reqId': '2504', 'url': '
18	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2507', 'url': 'https://it.youporn.com/favorites/'}}
19	{'comment': 'remove from favorite', 'flag': 'y', 'req': {'method': 'POST', 'params': {'video_id': ['360922']}, 'reqId': '2639', 'url': 'https://it.youporn.com/change/videos/360922/remove_favorites/'}}
20	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2642', 'url': 'https://it.youporn.com/myuploads/'}}
21	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2699', 'url': 'https://it.youporn.com/mycollections/'}}
22	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2753', 'url': 'https://it.youporn.com/collections/edit/1312281/'}}
23	{'comment': 'delete collection', 'flag': 'y', 'req': {'method': 'POST', 'params': {}, 'reqId': '2808', 'url': 'https://it.youporn.com/collections/delete/1312281/'}}
24	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2856', 'url': 'https://it.youporn.com/subscriptions/videos/'}}
25	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '2929', 'url': 'https://it.youporn.com/subscriptions/'}}
26	{'comment': '', 'flag': 'n', 'req': {'method': 'GET', 'params': {}, 'reqId': '3029', 'url': 'https://it.youporn.com/pornstar/21642/keisha-grey/'}}
27	{'comment': 'subscribe to channel', 'flag': 'y', 'req': {'method': 'POST', 'params': {'_token': ['2507819173276360704'], 'key': ['pornstar'], 'value': ['21642']}, 'reqId': '3118', 'url': 'https://it.youporn.com/
28	{'comment': 'unsubscribe from channel', 'flag': 'y', 'req': {'method': 'POST', 'params': {'_token': ['2507819173276360704'], 'key': ['pornstar'], 'value': ['21642']}, 'reqId': '3119', 'url': 'https://it.youporn.

Fig – 8.10 Mitch Detected Sites



Machine Learning Results

Method	Accuracy	Precesion	Recall
POST	0.8507604562737643	0.7838452787258248	0.8474784747847478
GET	0.8151140684410646	0.8769363166953529	0.8055335968379447

Fig – 8.11 Machine Learning Results

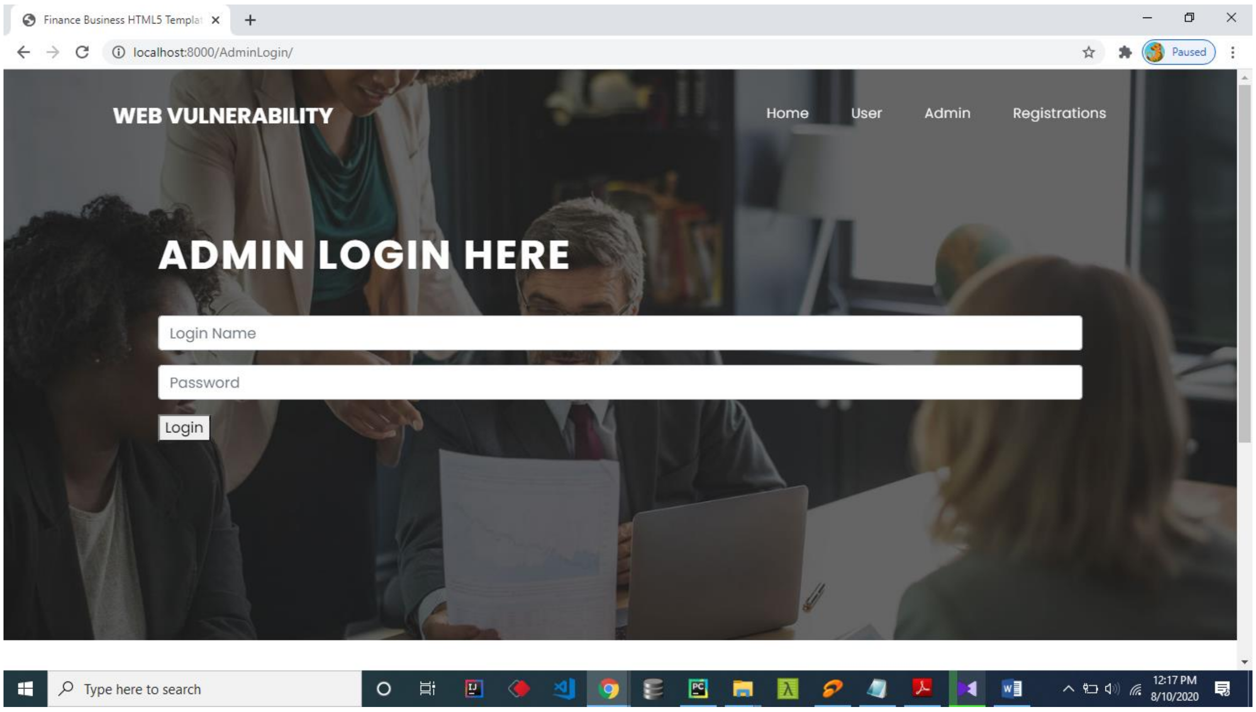
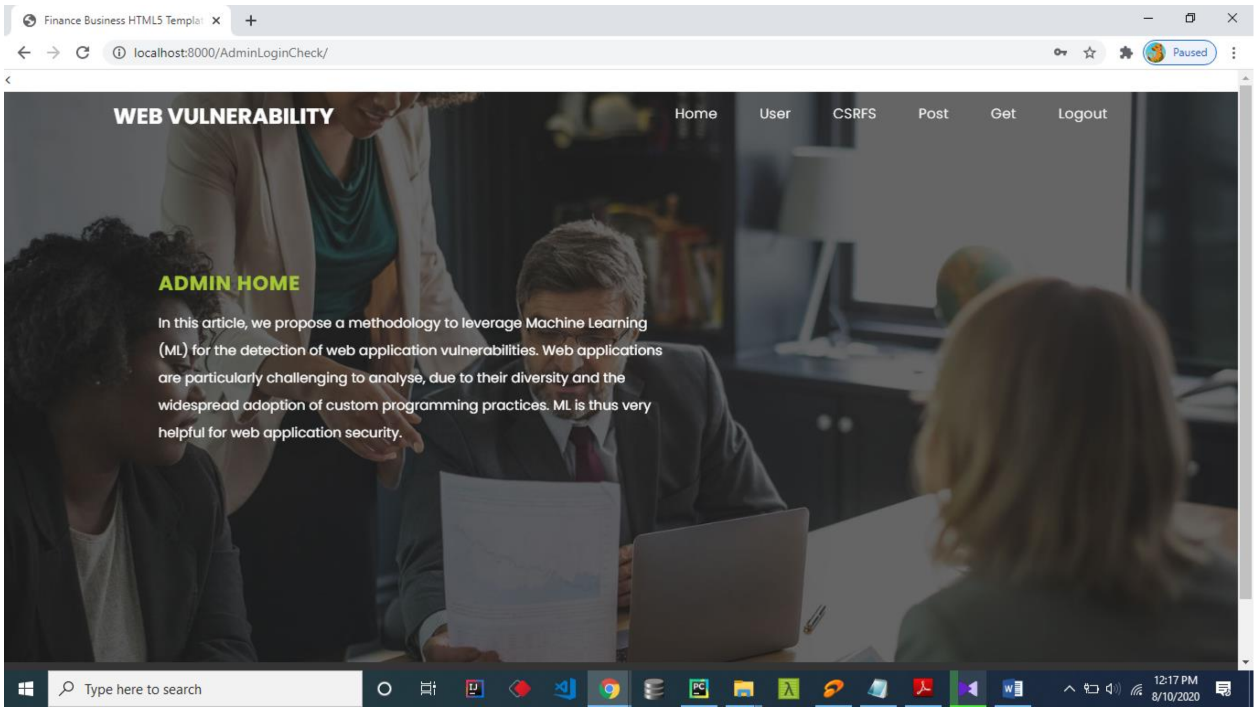


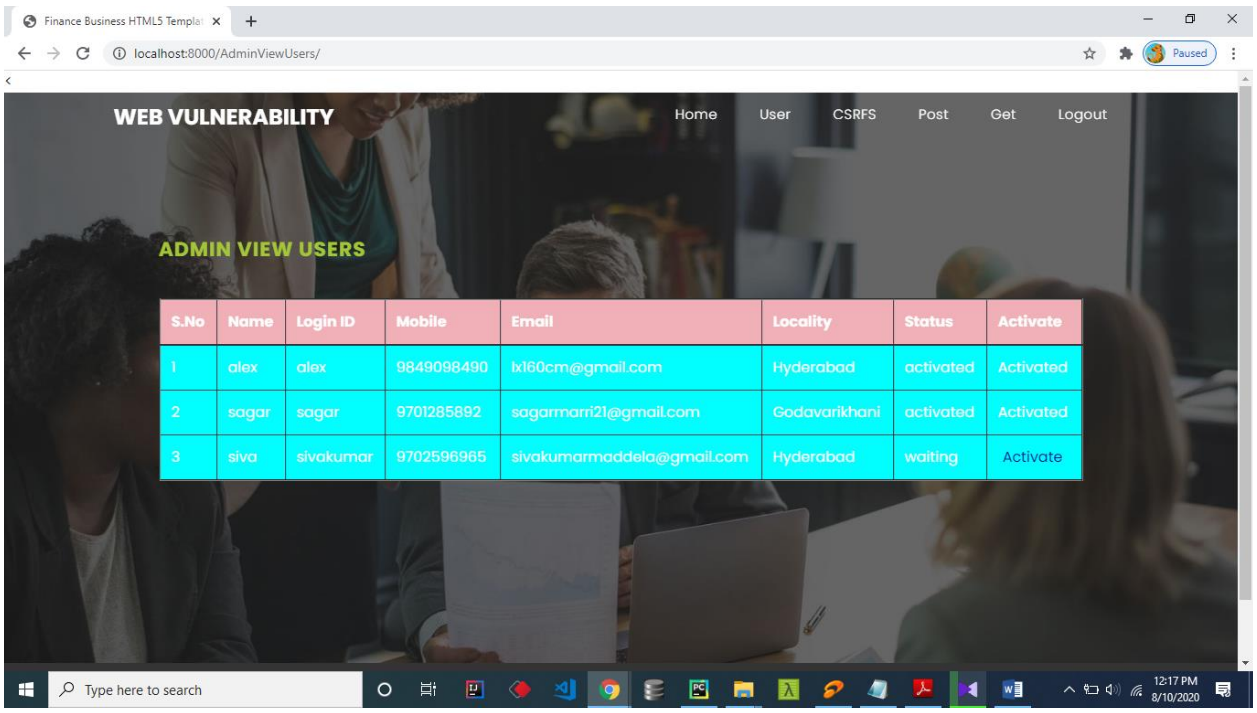
Fig – 8.12 Admin Login



ADMIN HOME

In this article, we propose a methodology to leverage Machine Learning (ML) for the detection of web application vulnerabilities. Web applications are particularly challenging to analyse, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful for web application security.

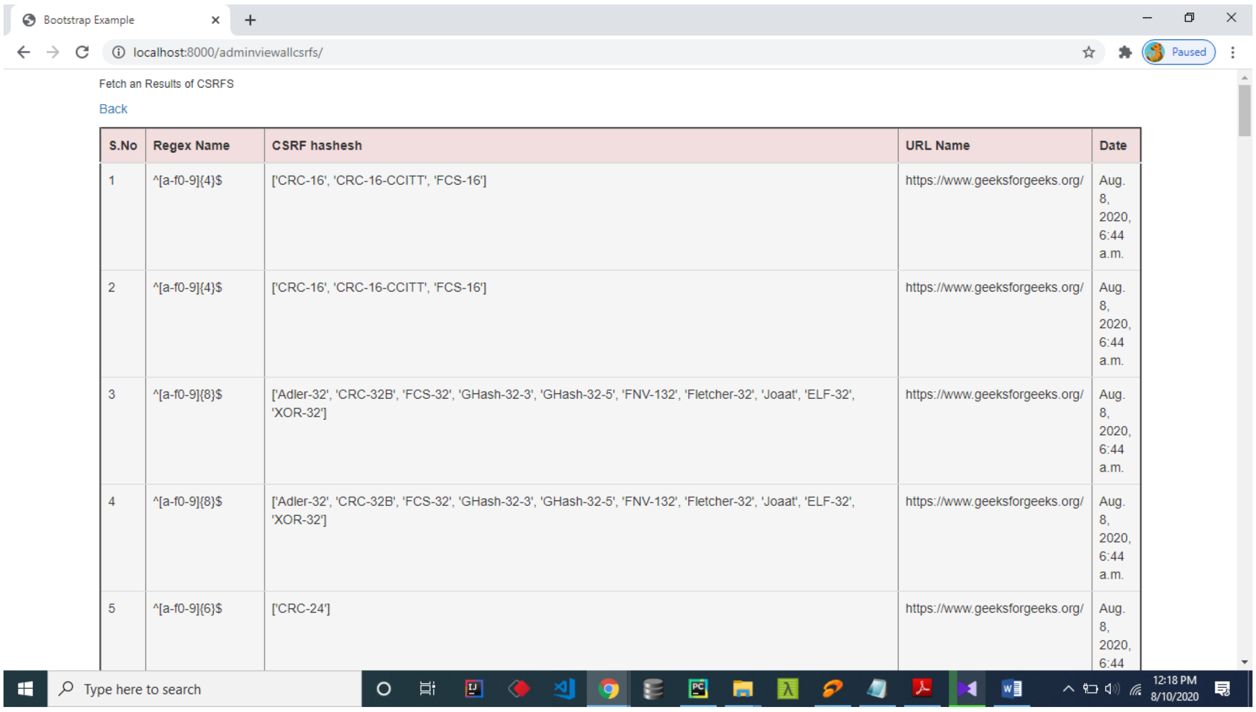
Fig – 8.13 Admin Home Page



ADMIN VIEW USERS

S.No	Name	Login ID	Mobile	Email	Locality	Status	Activate
1	alex	alex	9849098490	lx160cm@gmail.com	Hyderabad	activated	Activated
2	sagar	sagar	9701285892	sagarmarri21@gmail.com	Godavarikhani	activated	Activated
3	siva	sivakumar	9702596965	sivakumarmaddela@gmail.com	Hyderabad	waiting	Activate

Fig – 8.14 View Registered users:

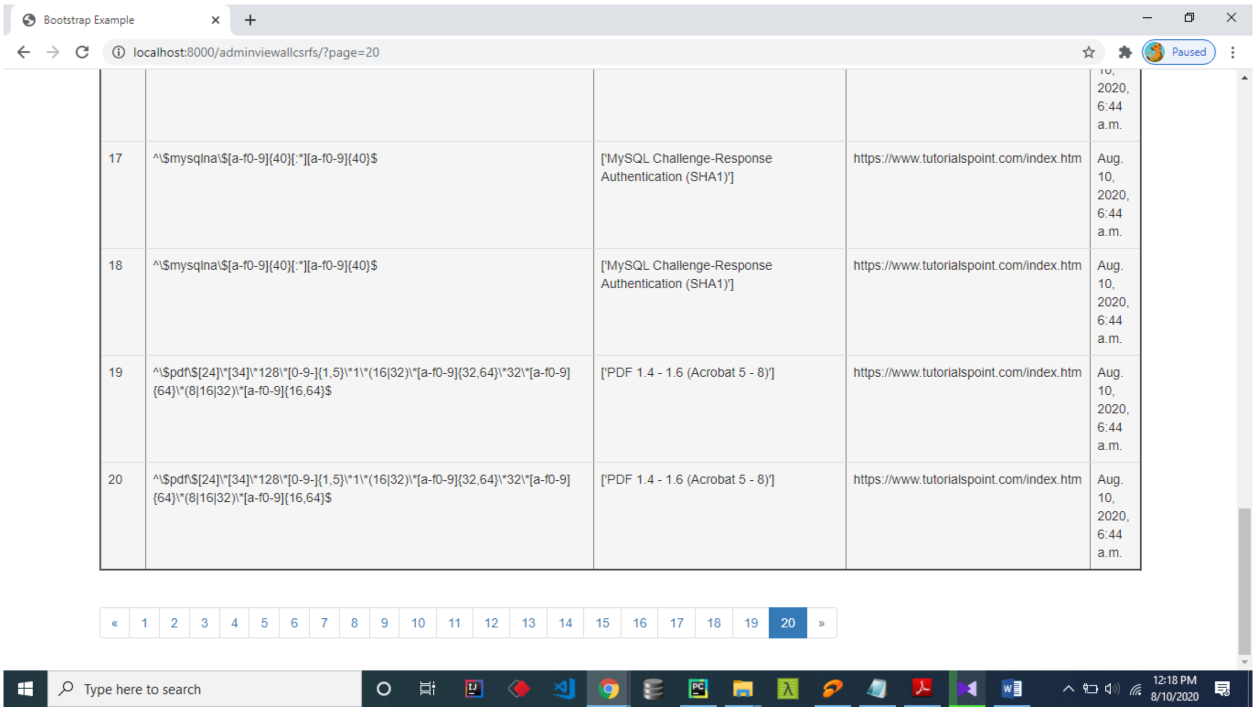


Fetch an Results of CSRFS

Back

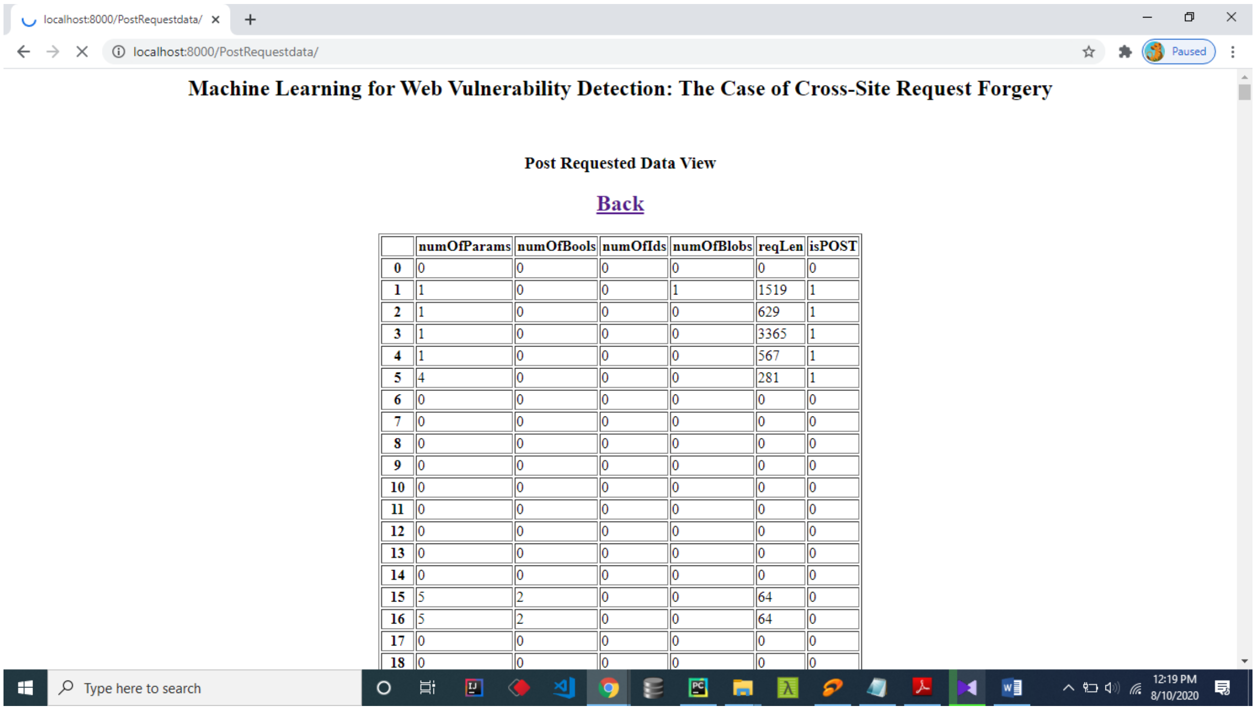
S.No	Regex Name	CSRF hashesh	URL Name	Date
1	^[a-f0-9]{4}$	['CRC-16', 'CRC-16-CCITT', 'FCS-16']	https://www.geeksforgeeks.org/	Aug. 8, 2020, 6:44 a.m.
2	^[a-f0-9]{4}$	['CRC-16', 'CRC-16-CCITT', 'FCS-16']	https://www.geeksforgeeks.org/	Aug. 8, 2020, 6:44 a.m.
3	^[a-f0-9]{8}$	['Adler-32', 'CRC-32B', 'FCS-32', 'GHash-32-3', 'GHash-32-5', 'FNV-132', 'Fletcher-32', 'Joaat', 'ELF-32', 'XOR-32']	https://www.geeksforgeeks.org/	Aug. 8, 2020, 6:44 a.m.
4	^[a-f0-9]{8}$	['Adler-32', 'CRC-32B', 'FCS-32', 'GHash-32-3', 'GHash-32-5', 'FNV-132', 'Fletcher-32', 'Joaat', 'ELF-32', 'XOR-32']	https://www.geeksforgeeks.org/	Aug. 8, 2020, 6:44 a.m.
5	^[a-f0-9]{6}$	['CRC-24']	https://www.geeksforgeeks.org/	Aug. 8, 2020, 6:44

Fig – 8.15 Admin View All CSRFS



				10, 2020, 6:44 a.m.
17	^\$mysqlna\$[a-f0-9]{40}[:*][a-f0-9]{40}$	['MySQL Challenge-Response Authentication (SHA1)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
18	^\$mysqlna\$[a-f0-9]{40}[:*][a-f0-9]{40}$	['MySQL Challenge-Response Authentication (SHA1)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
19	^\$pdf\$[24]*[34]*128*[0-9-]{1,5}*1*(16|32)*[a-f0-9]{32,64}*32*[a-f0-9]{64}*(8|16|32)*[a-f0-9]{16,64}$	['PDF 1.4 - 1.6 (Acrobat 5 - 8)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.
20	^\$pdf\$[24]*[34]*128*[0-9-]{1,5}*1*(16|32)*[a-f0-9]{32,64}*32*[a-f0-9]{64}*(8|16|32)*[a-f0-9]{16,64}$	['PDF 1.4 - 1.6 (Acrobat 5 - 8)']	https://www.tutorialspoint.com/index.htm	Aug. 10, 2020, 6:44 a.m.

Fig – 8.16 CSRFS


localhost:8000/PostRequestdata/

Machine Learning for Web Vulnerability Detection: The Case of Cross-Site Request Forgery

Post Requested Data View

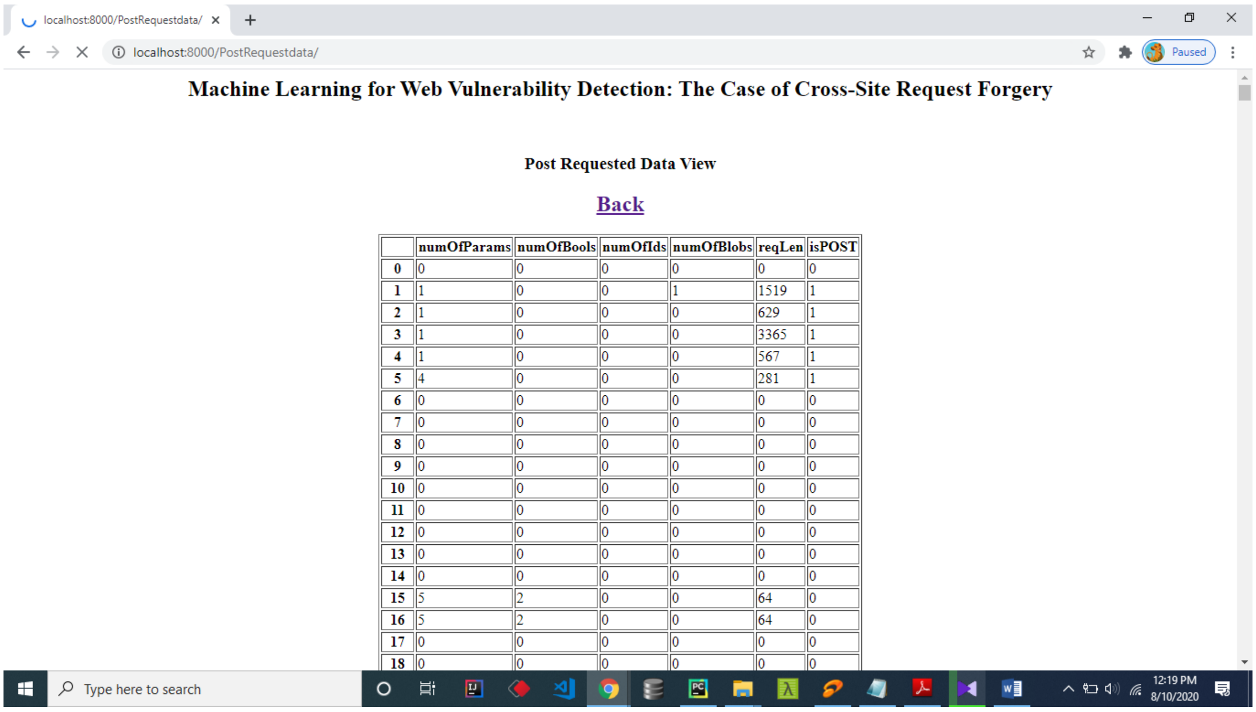
Back

	numOfParams	numOfBools	numOfIds	numOfBlobs	reqLen	isPOST
0	0	0	0	0	0	0
1	1	0	0	1	1519	1
2	1	0	0	0	629	1
3	1	0	0	0	3365	1
4	1	0	0	0	567	1
5	4	0	0	0	281	1
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	5	2	0	0	64	0
16	5	2	0	0	64	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0

Type here to search

12:19 PM 8/10/2020

Fig – 8.17 Post Data View


localhost:8000/PostRequestdata/

Machine Learning for Web Vulnerability Detection: The Case of Cross-Site Request Forgery

Post Requested Data View

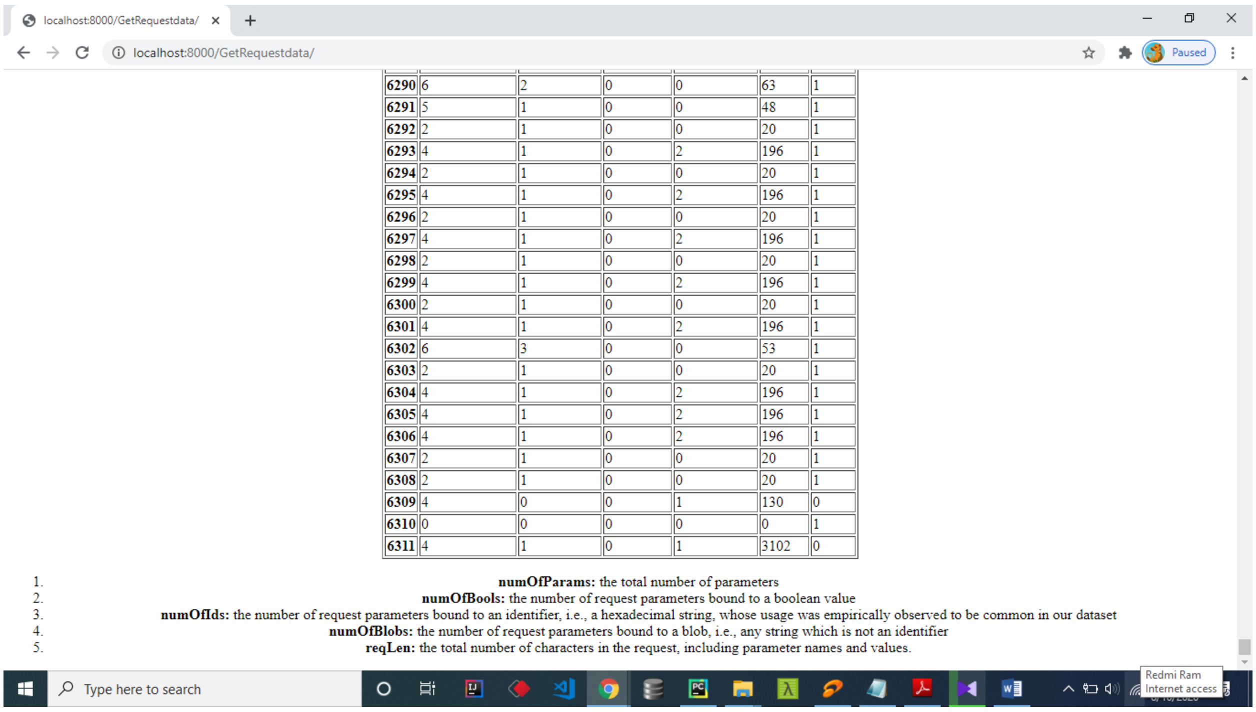
Back

	numOfParams	numOfBools	numOfIds	numOfBlobs	reqLen	isPOST
0	0	0	0	0	0	0
1	1	0	0	1	1519	1
2	1	0	0	0	629	1
3	1	0	0	0	3365	1
4	1	0	0	0	567	1
5	4	0	0	0	281	1
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	5	2	0	0	64	0
16	5	2	0	0	64	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0

Type here to search

12:19 PM 8/10/2020

Fig – 8.18 Get Data


localhost:8000/GetRequestdata/

6290	6	2	0	0	63	1
6291	5	1	0	0	48	1
6292	2	1	0	0	20	1
6293	4	1	0	2	196	1
6294	2	1	0	0	20	1
6295	4	1	0	2	196	1
6296	2	1	0	0	20	1
6297	4	1	0	2	196	1
6298	2	1	0	0	20	1
6299	4	1	0	2	196	1
6300	2	1	0	0	20	1
6301	4	1	0	2	196	1
6302	6	3	0	0	53	1
6303	2	1	0	0	20	1
6304	4	1	0	2	196	1
6305	4	1	0	2	196	1
6306	4	1	0	2	196	1
6307	2	1	0	0	20	1
6308	2	1	0	0	20	1
6309	4	0	0	1	130	0
6310	0	0	0	0	0	1
6311	4	1	0	1	3102	0

1. **numOfParams:** the total number of parameters
2. **numOfBools:** the number of request parameters bound to a boolean value
3. **numOfIds:** the number of request parameters bound to an identifier, i.e., a hexadecimal string, whose usage was empirically observed to be common in our dataset
4. **numOfBlobs:** the number of request parameters bound to a blob, i.e., any string which is not an identifier
5. **reqLen:** the total number of characters in the request, including parameter names and values.

Fig – 8.19 Attribute Descriptions

CHAPTER - 9
CONCLUSION

Web applications are particularly challenging to analyse, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful in the web setting, because it can take advantage of manually labeled data to expose the human understanding of the web application semantics to automated analysis tools. We validated this claim by designing Mitch, the first ML solution for the blackbox detection of CSRF vulnerabilities, and by experimentally assessing its effectiveness. We hope other researchers might take advantage of our methodology for the detection of other classes of web application vulnerabilities.

CHAPTER – 10

REFERENCES

1. Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta. Surviving the web: A journey into web session security. ACM Comput. Surv., 50(1):13:1–13:34, 2017.
2. Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli. Large-scale analysis & detection of authentication cross-site request forgeries. In 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017, pages 350–365, 2017.
3. Stefano Calzavara, Alvise Rabitti, Alessio Ragazzo, and Michele Bugliesi. Testing for integrity flaws in web sessions. In Computer Security - 24rd European Symposium on Research in Computer Security, ESORICS 2019, Luxembourg, Luxembourg, September 23-27, 2019, pages 606–624, 2019.
4. OWASP. OWASP Testing Guide. https://www.owasp.org/index.php/ OWASP Testing Guide v4 Table of Contents, 2016.
5. Jason Bau, Elie Bursztein, Divij Gupta, and John C. Mitchell. State of the art: Automated black-box web application vulnerability testing. In 31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA, pages 332–345, 2010.
6. Adam Doup´e, Marco Cova, and Giovanni Vigna. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In Detection of Intrusions and Malware, and Vulnerability Assessment, 7th International Conference, DIMVA 2010, Bonn, Germany, July 8-9, 2010. Proceedings, pages 111–131, 2010.
7. Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008, pages 75–88, 2008.
8. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of Machine Learning. The MIT Press, 2012.
9. Michael W. Kattan, Dennis A. Adams, and Michael S. Parks. A comparison of machine learning with human judgment. Journal of Management Information

Systems, 9(4):37–57, March 1993.

10. D. A. Ferrucci. Introduction to “This is Watson”. IBM Journal of Research and Development, 56(3):235–249, May 2012.
11. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489, Jan 2016.
12. Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. Cookiext: Patching the browser against session hijacking attacks. Journal of Computer Security, 23(4):509–537, 2015.
13. Stefano Calzavara, Gabriele Tolomei, Andrea Casini, Michele Bugliesi, and Salvatore Orlando. A supervised learning approach to protect client authentication on the web. TWEB, 9(3):15:1–15:30, 2015.
14. Stefano Calzavara, Mauro Conti, Riccardo Focardi, Alvise Rabitti, and Gabriele Tolomei. Mitch: A machine learning approach to the blackbox detection of CSRF vulnerabilities. In IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019, pages 528–543, 2019.
15. Giancarlo Pellegrino, Martin Johns, Simon Koch, Michael Backes, and Christian Rossow. Deemon: Detecting CSRF with dynamic analysis and property graphs. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 1757–1771, 2017.