



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

CSE3501 - Information Security Analysis and Audit

J Component Review 3

Topic:

Vulnerability Scanner

By

Rishika Agrawal– 20BIT0147

Riya Khatwani – 20BIT0410

Saniya Pendhari– 20BIT0057

Guided by

Dr. Jeyanthi N

SITE, VIT

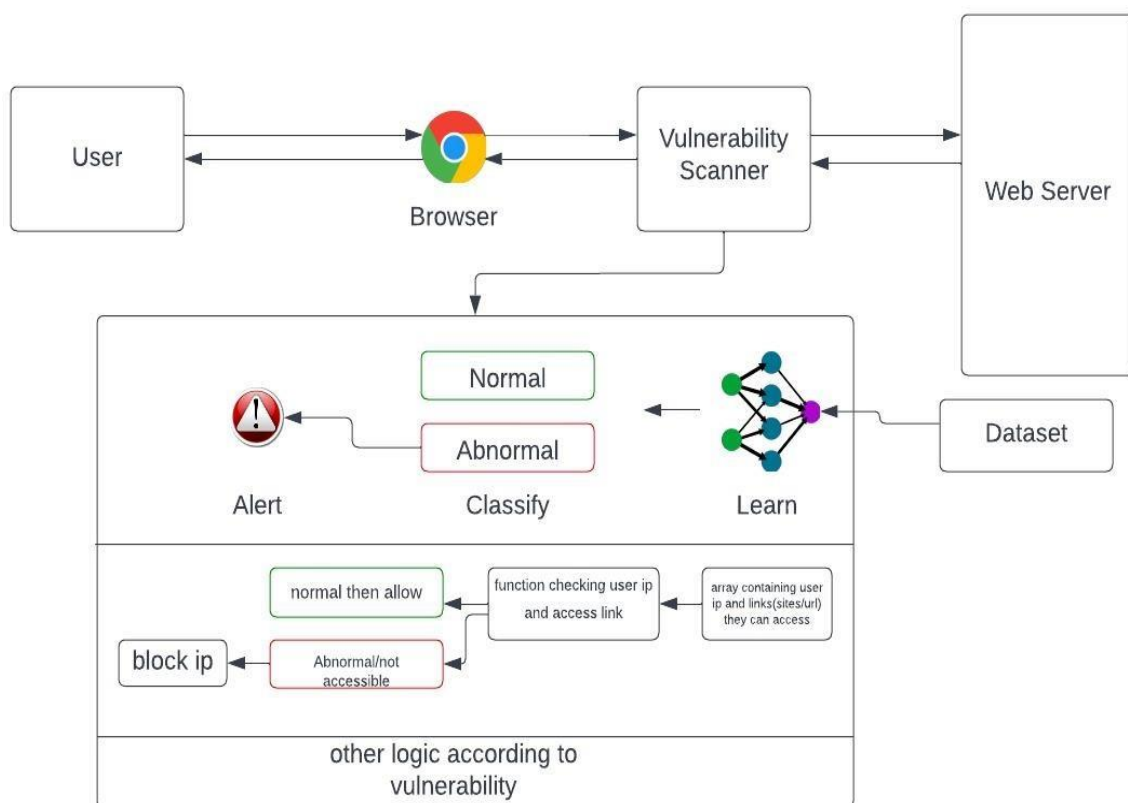
Problem Statement:

A vulnerability scanner enables organizations to monitor their networks, systems, and applications for security vulnerabilities.

Abstract:

Vulnerability scanners are used to bring to light security vulnerabilities in their computer systems, networks, applications and procedures. This project aims to explore a method that will enable the user to run a security check for their computer systems and networks. It takes help from techniques like artificial neural networks, other logics based on vulnerability and its pattern to help the user detect vulnerabilities in their system so that they can look for ways to combat them.

Architecture Diagram:






Vulnerability Scan on vit.ac.in through an online scanner




Opportunities & Experiments NEW

WebPageTest helps identify opportunities to improve a site's experience. Select one or more No-Code Experiments below and submit to test their impact.

Jump to Opportunities
by Category:

Is it Quick? ⚠ Not bad...
 8  8  13

Is it Usable? ⚠ Needs Improvement.
 4  4  4

Is it Resilient? ⚠ Needs Improvement.
 3  3  9

Is it Quick?

⚠ Not bad... This site took little time to connect and deliver initial code. It began rendering content with little delay. The largest contentful paint time was slower than ideal.

▼ Zero third-party fonts found.

Fonts on 3rd party domains may take longer to load due to DNS and connection steps that are not necessary when fonts are hosted on the same domain.

▼ Zero unused preloads were found.

Preloaded resources are fetched at a high priority, delaying the arrival of other resources in the page. In the case where a preloaded resource is never actually used by the page, that means potentially critical requests will be delayed, slowing down the initial loading of your site.

▼ Zero requests were found that resulted in an HTTP redirect.

HTTP redirects can result in additional DNS resolution, TCP connection and HTTPS negotiation times, making them very costly for performance, particularly on high latency networks.

▼ Potential SPOF: 1 3rd-party request is blocking page rendering.

By default, references to external JavaScript and CSS files will block the page from rendering. Third-party blocking requests are particularly risky, as your page's access relies on their response time and availability.

- <https://web-in21.mxraddon.com/t/Tracker.js>

▼ Several security vulnerabilities were detected by Snyk

Snyk has found 3 security vulnerabilities3 medium,

 Relevant Tips

Update the following JavaScript packages

- MEDIUM jquery 1.11.1 [View recommendation on Snyk](#) 
- MEDIUM jquery 1.11.1 [View recommendation on Snyk](#) 
- MEDIUM jquery 1.11.1 [View recommendation on Snyk](#) 

▼ Zero resources were found that were loaded over an Insecure connection.

Loading requests over HTTPS necessary for ensuring data integrity, protecting users personal information, providing core critical security, and providing access to many new browser features.

Literature Survey:

A Review on SQL Injection Vidushi, Prof. S. Niranjana Department of Computer Science & Engineering, Ganga Institute of Technology and Management, Kablana, Jhajjar, Haryana, India[1]

Malicious code can be introduced into entry fields via a technique called SQL Injection. This is one of the assault techniques hackers employ to obtain data from organisations. Database security is still an unsolved problem. SQL injection is a serious risk to our online application because it offers attackers unauthorized access to important database information. The SQL injection problem has been addressed by researchers and practitioners in a variety of ways, however existing solutions either fall short of addressing the entire breadth of the issue or have drawbacks that hinder their usage and acceptance. Only a small portion of the vast array of attack methods accessible to those attempting to exploit SQL injection flaws are known by many researchers and practitioners.

Malicious URL Detection: A Comparative Study
Shantanu Data, Janet B, Joshua Arul Kumar R [2]

One of the most frequent cybersecurity dangers is malicious websites, or malicious universal resource location (URL). Each year, they lose billions of rupees by hosting gratuitous material (spam, malware, unsuitable adverts, spoofing, etc.) and tempting naïve visitors to fall for scams (money loss, private information exposure, malware installation, extortion, phony shopping site, unexpected reward, etc.). Email, adverts, web searches, or connections from other websites can all encourage people to visit these websites. The user must click on the malicious URL in each instance. Due to the enormous number of data, changing patterns and technologies, as well as the complex relationships between characteristics, traditional classification approaches like blacklisting, regular expression, and signature matching approach are challenged. In this study, they analyse the binary classification issue of malicious URL identification and assess the performance of many popular machine learning classifiers. To train the model, we used a public dataset from Kaggle that had 450000 URLs. On the open phish website, harmful URLs were found using the best classifier. It was discovered to produce superior outcomes.

DeepXSS: Cross Site Scripting Detection Based on Deep Learning
Yong Fang, Yang Li, Liang Liu, Cheng Huang Sichuan University[3]

Cross-site scripting (XSS) attacks are one of the biggest vulnerabilities facing web applications today. According to polls done by the Open Web Applications Security Project, the XSS vulnerability has been among the top 10 Web application vulnerabilities ever since it became widely recognized (OWASP). Detecting and defending against XSS attacks is still one of the most crucial security concerns. In this research, we describe a unique deep learning-based method for detecting XSS assaults (called DeepXSS). First, we mapped each payload to a feature vector using word2vec to extract the feature of XSS payloads that collects word order information. Then, we used Long Short Term Memory (LSTM) recurrent neural networks to train and evaluate the detection model. According to experimental findings, the proposed deep learning-based XSS detection model successfully detects XSS assaults with an accuracy rate of 99.5% and a recall rate of 97.9% in actual datasets.

The vulnerabilities we have detected are

- 1) SQL INJECTION
- 2) PORT SCANNER
- 3) DDOS Attack detection
- 4) Malicious URL detection
- 5) XSS Detection

About the vulnerabilities-

SQL INJECTION-

SQL injection is a code injection technique that is used to execute SQL queries via the user input data to the vulnerable web application. It is one of the most common and dangerous web hacking techniques. A successful SQL injection exploit can cause a lot of harmful damage to the database and web application in general. For example, it can read sensitive data such as user passwords from the database, insert, modify and even delete data. We have taken all the html form information using soup package in python and extract important details from all the forms and then checked each of their methods and attributes if we can inject query in it or not to detect SQL injection vulnerability.

PORT SCANNER-

A port scan is a method for determining which ports on a network are open. As ports on a computer are the place where information is sent and received. Running a port scan on a network or server reveals which ports are open and listening (receiving information), as well as revealing the presence of security devices such as firewalls that are present between the sender and the target. This technique is known as fingerprinting. It is also valuable for testing network security and the strength of the system's firewall. Due to this functionality, it is also a popular reconnaissance tool for attackers seeking a weak point of access to break into a computer.

DDOS Attack detection-

A Distributed Denial of Service (DDoS) attack is an attempt to make an online service or a website unavailable by overloading it with huge floods of traffic generated from multiple sources. Unlike a Denial of Service (DoS) attack, in which one computer and one Internet connection is used to flood a targeted resource with packets, a DDoS attack uses many computers and many Internet connections, often distributed globally in what is referred to as a botnet. A large-scale volumetric DDoS attack can generate a traffic measured in tens of Gigabits (and even hundreds of Gigabits) per second.

Malicious URL detection

A malicious URL is a website link that is designed to promote virus attacks, phishing attacks, scams, and fraudulent activities. When a user clicks a malicious URL they can download computer viruses such as trojan horses, ransomware, worms, and spyware.

The end goal of these viruses is to access personal information, damage the user's device, and for financial gain. They may also destroy the company's network, leading to losses.

A malicious URL can also be used to lure people to submit their personal information on a fake website. This makes these people share their personal and sensitive information with unknown

people. They use the information for an ulterior motive. The harm caused by these malicious URLs can be very large. We have used data vectorization and logistic regression for this and got a good 96 percent accuracy for our algorithm to detect malicious URL. Our dataset for this contains all the URL and a label of good or bad according to different features which are used to decide their label.

XSS Detection

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise users' interactions with a vulnerable web application. The attacker aims to execute scripts in the victim's web browser by including malicious code on a normal web page. These flaws that allow these types of attacks are widespread in web applications with user input. We have taken all the html form information using soup package in python and extract important details from all the forms and then checked each of their methods and attributes if we can inject script in it or not to detect XSS vulnerability.

Code-

The full code implementation is in the following link-

https://colab.research.google.com/drive/1D4SmgCklaxfY_OrTZso_6nxQku-XMFsU?usp=sharing#scrollTo=KYmwZ_OlfdNm

Main part of code:

```
import os, time
import uuid
import hashlib

import requests
from pprint import pprint
from bs4 import BeautifulSoup as bs
from urllib.parse import urljoin

import pyfiglet
import sys
import socket
from datetime import datetime

import requests
from bs4 import BeautifulSoup as bs
from urllib.parse import urljoin
from pprint import pprint

import socket
import struct

from datetime import datetime

# initialize an HTTP session & set the browser
s = requests.Session()
s.headers["User-Agent"] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
```

```
(KHTML, like Gecko) Chrome/83.0.4103.106 Safari/537.36"
```

```
def get_all_forms(url):
    """Given a `url`, it returns all forms from the HTML content"""
    soup = bs(s.get(url).content, "html.parser")
    return soup.find_all("form")

def get_form_details(form):
    """
    This function extracts all possible useful information about an HTML `form`
    """
    details = {}
    # get the form action (target url)
    try:
        action = form.attrs.get("action").lower()
    except:
        action = None
    # get the form method (POST, GET, etc.)
    method = form.attrs.get("method", "get").lower()
    # get all the input details such as type and name
    inputs = []
    for input_tag in form.find_all("input"):
        input_type = input_tag.attrs.get("type", "text")
        input_name = input_tag.attrs.get("name")
        input_value = input_tag.attrs.get("value", "")
        inputs.append({"type": input_type, "name": input_name, "value":
input_value})
    # put everything to the resulting dictionary
    details["action"] = action
    details["method"] = method
    details["inputs"] = inputs
    return details

def is_vulnerable(response):
    """A simple boolean function that determines whether a page
    is SQL Injection vulnerable from its `response`"""
    errors = {
        # MySQL
        "you have an error in your sql syntax;",
        "warning: mysql",
        # SQL Server
        "unclosed quotation mark after the character string",
        # Oracle
        "quoted string not properly terminated",
    }
    for error in errors:
        # if you find one of these errors, return True
        if error in response.content.decode().lower():
            return True
    # no error detected
    return False

def scan_sql_injection(url):
```

```

# test on URL
for c in "\"'":
    # add quote/double quote character to the URL
    new_url = f"{url}{c}"
    print("[!] Trying", new_url)
    # make the HTTP request
    res = s.get(new_url)
    if is_vulnerable(res):
        # SQL Injection detected on the URL itself,
        # no need to proceed for extracting forms and submitting them
        print("[+] SQL Injection vulnerability detected, link:", new_url)
        return

# test on HTML forms
forms = get_all_forms(url)
print(f"[+] Detected {len(forms)} forms on {url}.")
for form in forms:
    form_details = get_form_details(form)
    for c in "\"'":
        # the data body we want to submit
        data = {}
        for input_tag in form_details["inputs"]:
            if input_tag["type"] == "hidden" or input_tag["value"]:
                # any input form that is hidden or has some value,
                # just use it in the form body
                try:
                    data[input_tag["name"]] = input_tag["value"] + c
                except:
                    pass
            elif input_tag["type"] != "submit":
                # all others except submit, use some junk data with special cha
racter

                data[input_tag["name"]] = f"test{c}"
        # join the url with the action (form request URL)
        url = urljoin(url, form_details["action"])
        if form_details["method"] == "post":
            res = s.post(url, data=data)
        elif form_details["method"] == "get":
            res = s.get(url, params=data)
        # test whether the resulting page is vulnerable
        if is_vulnerable(res):
            print("[+] SQL Injection vulnerability detected, link:", url)
            print("[+] Form:")
            pprint(form_details)
            break

#DDOS Attack

def ddosattack():
    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, 8)

    dict = {}
    file_txt = open("attack_DDoS.txt", 'a')

```



```

t1 = str(datetime.now())

file_txt.writelines(t1)
file_txt.writelines("\n")

No_of_IPs = 15
R_No_of_IPs = No_of_IPs +10

while True:
    pkt = s.recvfrom(2048)
    ipheader = pkt[0][14:34]
    ip_hdr = struct.unpack("!8sB3s4s4s",ipheader)
    IP = socket.inet_ntoa(ip_hdr[3])
    print ("The Source of the IP is:", IP)

if dict.has_key(IP):
    dict[IP] = dict[IP]+1
    print (dict[IP])

if(dict[IP] > No_of_IPs) and (dict[IP] < R_No_of_IPs) :
    line = "DDOS attack is Detected: "
    file_txt.writelines(line)
    file_txt.writelines(IP)
    file_txt.writelines("\n")
else:
    dict[IP] = 1

#XSS Detection

def get_all_forms(url):
    """Given a `url`, it returns all forms from the HTML content"""
    soup = bs(requests.get(url).content, "html.parser")
    return soup.find_all("form")

def get_form_details(form):
    """
    This function extracts all possible useful information about an HTML `form`
    """
    details = {}
    # get the form action (target url)
    action = form.attrs.get("action", "").lower()
    # get the form method (POST, GET, etc.)
    method = form.attrs.get("method", "get").lower()
    # get all the input details such as type and name
    inputs = []
    for input_tag in form.find_all("input"):
        input_type = input_tag.attrs.get("type", "text")
        input_name = input_tag.attrs.get("name")
        inputs.append({"type": input_type, "name": input_name})
    # put everything to the resulting dictionary
    details["action"] = action
    details["method"] = method
    details["inputs"] = inputs

```

```

return details

def submit_form(form_details, url, value):
    """
    Submits a form given in `form_details`
    Params:
        form_details (list): a dictionary that contain form information
        url (str): the original URL that contain that form
        value (str): this will be replaced to all text and search inputs
    Returns the HTTP Response after form submission
    """
    # construct the full URL (if the url provided in action is relative)
    target_url = urljoin(url, form_details["action"])
    # get the inputs
    inputs = form_details["inputs"]
    data = {}
    for input in inputs:
        # replace all text and search values with `value`
        if input["type"] == "text" or input["type"] == "search":
            input["value"] = value
        input_name = input.get("name")
        input_value = input.get("value")
        if input_name and input_value:
            # if input name and value are not None,
            # then add them to the data of form submission
            data[input_name] = input_value

    print(f"[+] Submitting malicious payload to {target_url}")
    print(f"[+] Data: {data}")
    if form_details["method"] == "post":
        return requests.post(target_url, data=data)
    else:
        # GET request
        return requests.get(target_url, params=data)

def scan_xss(url):
    """
    Given a `url`, it prints all XSS vulnerable forms and
    returns True if any is vulnerable, False otherwise
    """
    # get all the forms from the URL
    forms = get_all_forms(url)
    print(f"[+] Detected {len(forms)} forms on {url}.")
    js_script = "<Script>alert('hi')</script>"
    # returning value
    is_vulnerable = False
    # iterate over all forms
    for form in forms:
        form_details = get_form_details(form)
        content = submit_form(form_details, url, js_script).content.decode()
        if js_script in content:
            print(f"[+] XSS Detected on {url}")
            print(f"[*] Form details:")

```

```

        pprint(form_details)
        is_vulnerable = True
        # won't break because we want to print available vulnerable forms
    return is_vulnerable

```

Input Menu

```

def Menu():
    print ("Please Select the mode of operation:- \n")
    print ("1) SQL INJECTION")
    print ("2) PORT SCANNER")
    print ("3) DDOS Attack detection")
    print ("4) Malicious URL detection")
    print ("5) XSS Detection")
    print ("6) Exit the program :[\n")

```

#ACTUAL PROGRAM

```

if __name__ == '__main__':
    die =1
    try:
        while(die):
            os.system("cls")
            Menu()
            choice = int(input("Enter your choice: "))
            print("\n")
            if choice == 1:
                print("##### SQL INJECTION #####")
                url = input("\nPlease enter URL TO CHECK ")
                #http://testphp.vulnweb.com/artists.php?artist=1

                scan_sql_injection(url)
                time.sleep(10)
            elif choice == 2:
                print("PORT SCANNER #####")
                target= input("\nPlease enter target")
                #104.31.85.168

            try:

                # will scan ports between 1 to 65,535
                for port in range(1,6):
                    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    socket.setdefaulttimeout(1)

                    # returns an error indicator

```

```

        result = s.connect_ex((target,port))
        if result ==0:
            print("Port {} is open".format(port))
        else:
            print("no ports open")
        s.close()

    except KeyboardInterrupt:
        print("\n Exiting Program !!!!")
        sys.exit()
    except socket.gaierror:
        print("\n Hostname Could Not Be Resolved !!!!")
        sys.exit()
    except socket.error:
        print("\ Server not responding !!!!")
        sys.exit()

    time.sleep(1)
elif choice == 3:
    print("##### DDOS attack detection #####")
    ddosattack();
    time.sleep(10)
elif choice == 4:
    print("##### Malicious URL detection #####")
    input_string = input('Enter url of a list separated by , ')
    print("\n")
    X_predict = input_string.split(",")

    X_predict = vectorizer.transform(X_predict)
    New_predict = logit.predict(X_predict)
    print(New_predict)

    time.sleep(10)
elif choice == 5:
    print("##### XSS Detection #####")
    if __name__ == "__main__":
        url = input("\nPlease enter URL TO CHECK ")
        #https://xss-game.appspot.com/level1/frame
        print(scan_xss(url))
    time.sleep(10)
elif choice == 6:
    die = 0
    quit()
else:
    print("This choice is not in the menu. Try Again!!!")
    die=1
except KeyboardInterrupt:
    print("\n\nInterrupt received! Exiting cleanly...\n")

```

Output-

```
▶ urls_data.head()
```

	url	label
0	diaryofagameaddict.com	bad
1	espdesign.com.au	bad
2	iamagameaddict.com	bad
3	kalantzis.net	bad
4	slightlyoffcenter.net	bad

```
[13] # Accuracy of Our Model
      print("Accuracy ",logit.score(X_test, y_test))

      Accuracy  0.9619468921313308
```

```

Please Select the mode of operation:-

...
1) SQL INJECTION
2) PORT SCANNER
3) DDOS Attack detection
4) Malicious URL detection
5) XSS Detection
6) Exit the program :[.

Enter your choice: 1

##### SQL INJECTION #####

Please enter URL TO CHECK http://testphp.vulnweb.com/artists.php?artist=1
[!] Trying http://testphp.vulnweb.com/artists.php?artist=1"
[+] SQL Injection vulnerability detected, link: http://testphp.vulnweb.com/artists.php?artist=1"
Please Select the mode of operation:-
```

```
... Please Select the mode of operation:-
```

- 1) SQL INJECTION
- 2) PORT SCANNER
- 3) DDOS Attack detection
- 4) Malicious URL detection
- 5) XSS Detection
- 6) Exit the program :[.

Enter your choice: 2

PORT SCANNER #####

Please enter target104.31.85.168

no ports open

no ports open

no ports open

no ports open

no ports open

Please Select the mode of operation:-

- 1) SQL INJECTION
- 2) PORT SCANNER
- 3) DDOS Attack detection
- 4) Malicious URL detection
- 5) XSS Detection
- 6) Exit the program :[.

Enter your choice: 3

```
##### DDOS attack detection #####
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 172.28.0.3
The Source of the IP is: 172.28.0.3
The Source of the IP is: 172.28.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 127.0.0.1
The Source of the IP is: 172.28.0.3
```

attack_DDoS.txt X

```
1 2022-11-18 14:13:39.554150
2
```

```
Please Select the mode of operation:-
1) SQL INJECTION
2) PORT SCANNER
3) DDOS Attack detection
4) Malicious URL detection
5) XSS Detection
6) Exit the program :[.

Enter your choice: 4

##### Malicious URL detection #####
Enter url of a list separated by , https://www.section.io/engineering-education/, https://www.youtube.com/, https://www.traversymedia.com/, https://www.kleinhundezuhause.com , http://t

['good' 'good' 'bad' 'bad' 'bad' 'bad']
Please Select the mode of operation:-
1) SQL INJECTION
2) PORT SCANNER
3) DDOS Attack detection
4) Malicious URL detection
5) XSS Detection
6) Exit the program :[.

Enter your choice: 5

##### XSS Detection #####
```

Enter your choice: 5

XSS Detection

```
Please enter URL TO CHECK https://xss-game.appspot.com/level1/frame
[+] Detected 1 forms on https://xss-game.appspot.com/level1/frame.
[+] Submitting malicious payload to https://xss-game.appspot.com/level1/frame
[+] Data: {'query': "<Script>alert('hi')</scripT>"}
[+] XSS Detected on https://xss-game.appspot.com/level1/frame
[*] Form details:
{'action': '',
 'inputs': [{'name': 'query',
              'type': 'text',
              'value': "<Script>alert('hi')</scripT>"},
            {'name': None, 'type': 'submit'}],
 'method': 'get'}
True
```

Conclusion:

Through this project we have tried to detect different vulnerabilities using different methods like data vectorization, logistic regression and many more and we have developed a vulnerability scanner which can scan and detect 5 vulnerabilities with taking input from user that is SQL injection, port scanner, malicious URL, DDOS attack and XSS.

References:

1. A Review on SQL Injection Vidushi, Prof. S. Niranjana Department of Computer Science & Engineering, Ganga Institute of Technology and Management, Kablana, Jhajjar, Haryana, India
2. Malicious URL Detection: A Comparative Study
Shantanu Data, Janet B, Joshua Arul Kumar R
3. DeepXSS: Cross Site Scripting Detection Based on Deep Learning
Yong Fang, Yang Li, Liang Liu, Cheng Huang Sichuan University