ADVANCED DETECTION AND MITIGATION OF CROSS-SITE SCRIPTING ATTACKS THROUGH INTELLIGENT JAVASCRIPT CODE INJECTION ANALYSIS

Ashmit Gupta
School of Computer Science and Engineering Vellore Institute of Technology, Vellore, India
Student Member IEEE

Neeraj Srivastava
Associate Professor
Department of Electronics & Communication Engineering Rustamji Institute of Technology, BSF Academy Tekanpur Gwalior, India
Senior Member IEEE

Ajit Kumar Srivastava
Associate Professor & Head Department of CSESISTec-R, Bhopal , India

ABSTRACT

Cross-site scripting attack, abbreviated as XSS , has been an incessant problem for Web applications since the early 2000s. It is a code injection attack on the client -side where an attacker injects malicious payload into a vulnerable web application, without client's knowledge.

The attacker is often successful in executing the malicious code inadvertently in the browser of an unwary user. Attempts have been made to implement the detection of XSS attacks using Genetic Algorithm, Web Vulnerabilities Finder, Fuzzy Interference Model, but they all come with drawbacks. Implementation URL of the website is collected through an extension and vulnerability is checked by injecting java-script code to the website. If the website is vulnerable, then display a pop -up stating "Website is vulnerable, be aware", else display a pop -up stating "Website is not vulnerable". It is a low cost model which is easy to implement.

Keywords- XSS , FI Model, Genetic Algorithm, Web Vulnerabilities Finder, Cyber Attack

INTRODUCTION

Web applications and websites are becoming more and more widespread as a result of increased internet usage, which has also led to an increase in cyber attacks on web applications and websites. XSS (Cross -Site Scripting) attacks are one of the most prevalent forms of cyber attack on web applications and websites out of all the other sorts of attacks.

Cross Site Scripting (XSS) is vulnerability in a web application that allows a third party to execute a script in the user's browser on behalf of the web application. Malicious scripts are injected into websites that are normally safe and reputable in Cross-Site Scripting (XSS) attacks. XSS attacks take place when an attacker sends malicious code, typically in the form of a browser side script, to a separate end user using an online application. It allows an attacker to by-pass the origin policy that is designed to segregate different websites from each other.

An attacker who exploits this vulnerability can:

- Read any data that the user is able to access
- Capture user's login credentials
- Impersonate or masquerade as the victim user.

Cross-site scripting attacks enable attackers to inject client -side scripts into web pages viewed by the user. The scripts are executed automatically without user’s consent, unless manually disabled. Besides, the users don’t seem to care about configuring the browsers securely. The number of attacks on users, by exploiting the browser vulnerabilities, has risen at alarming rate.

Already existing attack detecting mechanisms have failed miserably in most scenarios. Additionally, people haven't paid much attention to configuring their browsers securely with the help of accessible plug-ins and extensions.

The users have to protect themselves from these vulnerabilities and hence the detection of Cross-site scripting attacks is very important .to effectively detect and defend XSS attacks are still one of the most important security issues. Therefore, we need better solutions/ methods for detection of Cross-site Scripting (XSS) attacks.

LITERATURE SURVEY

Zhonglin Liu, Yong Fang, Cheng Huang and Yijia Xu (2022) - Genetic Algorithm

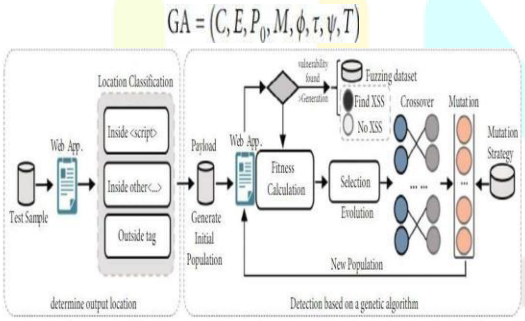


FIGURE 2: The system architecture of GAXSS.

- ❑ Genetic algorithm starts from an initial population, through random selection, crossover and mutation.
- ❑ The algorithm generates a group of individuals more suitable for the environment so that the group evolves to better areas in the exploration space.
- ❑ In the process of exploration, obtaining the local optimal solution is not easy.

Drawback : Genetic algorithm is random and hence a large amount of time is consumed in detecting vulnerabilities.

Muhammad Noman Khalid, Muhammad Iqbal, Kamran Rasheed, Malik MuneebAbid (2020) - Web Vulnerabilities Finder (WVF)

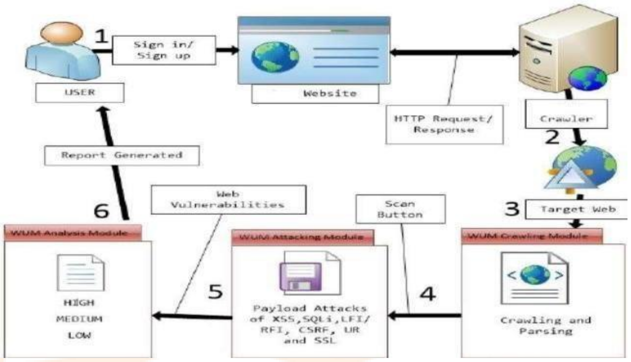


Fig.1. The Architecture of WVF scanner

- ❑ Web Vulnerabilities Finder, abbreviated as WVF , is a scanning tool capable of performing efficient penetration tests on php and websites with “.net” domain to identify web vulnerabilities.
- ❑ It is capable of finding hidden SQLi and XSS vulnerabilities.
- ❑ It comprises of : Crawling Module, Attack Module and Analysis Module

Drawback : WVF sometimes generates false positives and false negatives.

Bakare K. Ayeni, Junaidu B. Sahalu and Kolawole R. Adeyanju (2020)- Fuzzy based

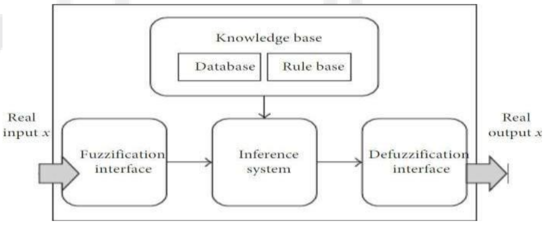


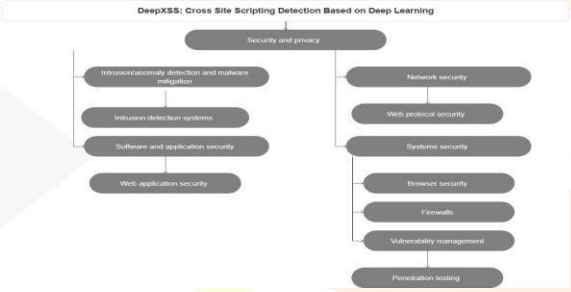
FIGURE 2: Mamdani fuzzy inference system

Fuzzy-based approach contributes to :

- ❑ Selection and implementation of DOM-based features for XSS detection.
- ❑ Application of the fuzzy logic inference system to detect vulnerabilities in webapplications.
- ❑ Implementation of the UI that gives users an idea about the level of exposure to XSS attacks while visiting a website.

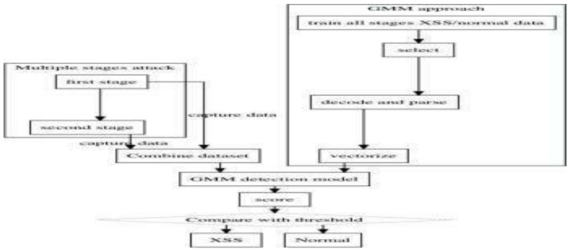
Drawback : In the Fuzzy Interference Model, many important categories of vulnerabilities are triggered by unexpected user inputs and can appear anywhere within the application.

Jingchi Zhang, Yu-Tsern Jou, Xiangyang Li (2019)


DeepXSS: Cross Site Scripting Detection Based on Deep Learning
Security and privacy
Intrusion/anomaly detection and malware mitigation
Intrusion detection systems
Software and application security
Web application security
Network security
Web protocol security
Systems security
Browser security
Firewalls
Vulnerability management
Penetration testing

- ❑ A probabilistic model, Gaussian mixing model, posits that all of the data points were produced by combining a limited number of Gaussian distributions with unknowable parameters.
- ❑ The expectation maximization (EM) approach for fitting a mixture of Gaussian models is implemented by the Gaussian Mixture object.
- ❑ A Gaussian Mixture Model can be learned from train data using theGaussianMixture.fit technique.
- ❑ Using the Gaussian Mixture predict technique and test data, it may assign each samplethe Gaussian that it most likely belongs to.

Yong Fang, Yang Li , Liang Liu,Cheng Huang(2021) – DeepXSS



- ❑ Deep XSS is to detect XSS attacks based on deep learning.
- ❑ It uses word2vec in order to extract the word order information from XSS payloads and map each payload to a feature vector.
- ❑ According to experimental findings, the proposed deep learning-based XSS detection model obtains a precision rate of 99.5% and a recall rate of 97.9 % in real datasets.

PROPOSED WORK

All of the XSS prevention tools come into scenario, when an XSS attack is being performed on a website or has been successfully executed. The draw backs of this are:

- ❑ Loss on money
- ❑ Loss of time
- ❑ Loss of resources
- ❑ Dent to company's reputation
- ❑ Loss of data
- ❑ Threat to integrity and confidentiality

Approach (What Did We Do?)

- ❑ So, you can call our approach as "prevention is better than cure".
- ❑ Basically, our approach is to make a XSS detection extension which can be added to any browser and it identifies all the forms from that web page using web scrapping and get all those forms which has input type as text and submit method as post or get requests.
- ❑ Now we will inject a random query into the inputs.
- ❑ As we know that for every request to the server there is a response from it.
- ❑ If the response is negative then we can say that the website is not vulnerable.
- ❑ If the response is positive, it means that the website is

vulnerable and we will displayto the user, the exact form in which there is vulnerability with parameters including:

-Request method

-Input type

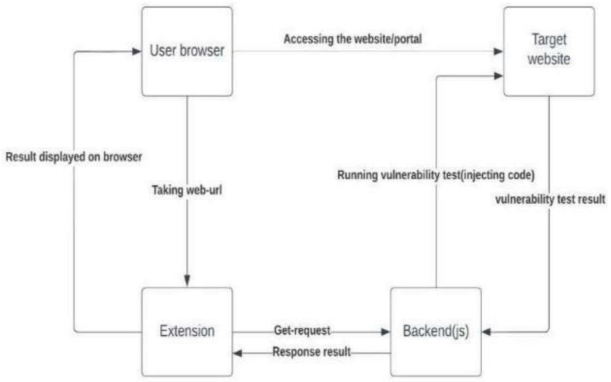
-Response from server

METHODOLOGY

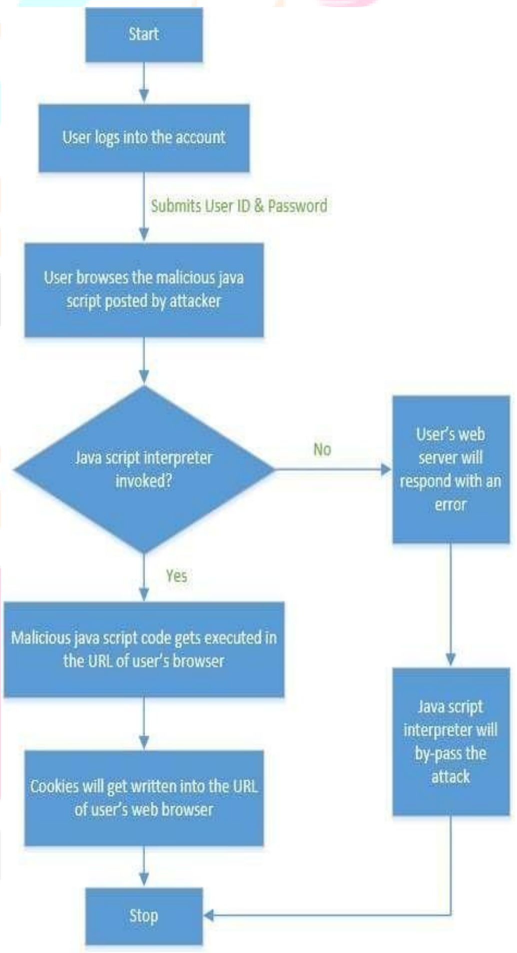
We have build an extension as a json file and have implemented detection of XSS attack using python and node.js code. Using the extension we find out all the inputs types and check if it contains malicious code. If yes then we print the malicious sites in the extension window. This will tell the user that this website is malicious and the user will not fill any details in the website. In this way our extension will prevent the user from getting attacked by cross site scripting attack.

1. User has to login to the account.
2. User has to download and install XSS _Vulnerability_Extension on the web browser.
3. User must open any webpage after installation of the extension.
4. Extension captures the URL of currently opened tab.
5. URL is sent to the created backend.
6. URL is extracted in the backend and is passed to the scanner python file.
7. The Beautiful Soup library in the python code detects forms on the webpage.
8. A dummy java-script is created to inject into the webpage.
9. All the forms present on the webpage are traversed thoroughly.
10. Form details like their action, method and inputs are extracted.
11. Form is submitted using all the collected details and java-script is inserted in the textinput field.
12. If the script gets inserted to the webpage then, it is vulnerable, else it is not vulnerable.

BLOCK DIAGRAM


User browser
Accessing the website/portal
Target website
Result displayed on browser
Taking web-url
Running vulnerability test(injecting code)
vulnerability test result
Extension
Get-request
Response result
Backend(js)

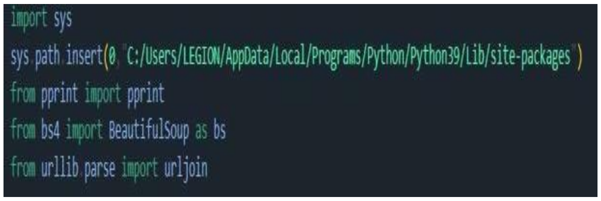
FLOW CHART


Start
User logs into the account
Submits User ID & Password
User browses the malicious java script posted by attacker
Java script interpreter invoked?
No
User's web server will respond with an error
Yes
Malicious java script code gets executed in the URL of user's browser
Java script interpreter will by-pass the attack
Cookies will get written into the URL of user's web browser
Stop

IMPLEMENTATION

Python script for detecting the vulnerability

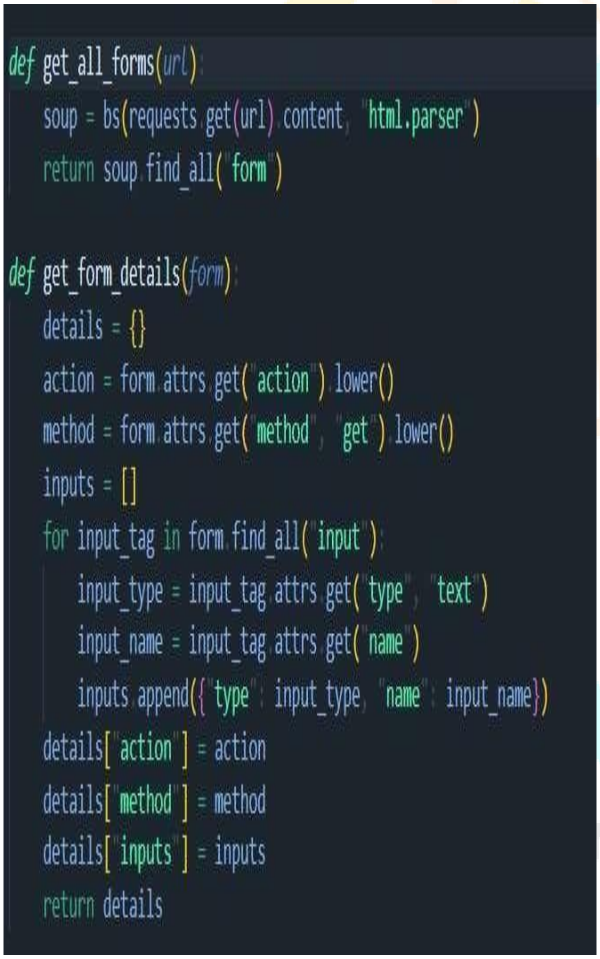
- **Libraries needed**



```
import sys
sys.path.insert(0, "C:/Users/LEGION/AppData/Local/Programs/Python/Python39/Lib/site-packages")
from pprint import pprint
from bs4 import BeautifulSoup as bs
from urllib.parse import urljoin
```

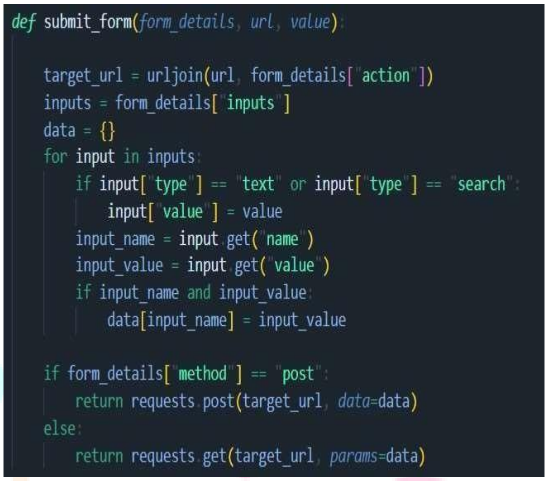
Set the path to the directory where all paths have been downloaded.

- **Getting the forms**



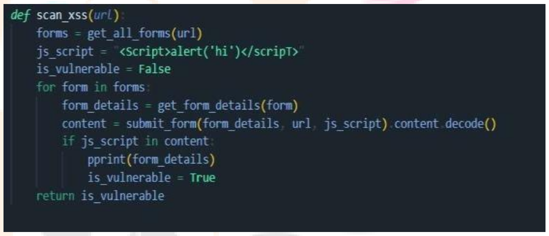
```
def get_all_forms(url):
    soup = bs(requests.get(url).content, "html.parser")
    return soup.find_all("form")

def get_form_details(form):
    details = {}
    action = form.attrs.get("action").lower()
    method = form.attrs.get("method", "get").lower()
    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})
    details["action"] = action
    details["method"] = method
    details["inputs"] = inputs
    return details
```

Select the forms having input type as text and methods as POST and GET



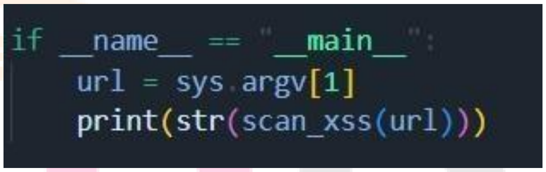
```
def submit_form(form_details, url, value):

    target_url = urljoin(url, form_details["action"])
    inputs = form_details["inputs"]
    data = {}
    for input in inputs:
        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:
            data[input_name] = input_value

    if form_details["method"] == "post":
        return requests.post(target_url, data=data)
    else:
        return requests.get(target_url, params=data)
```

Injecting the JS query into the inputs and posting it



```
def scan_xss(url):
    forms = get_all_forms(url)
    js_script = "<Script>alert('hi')</scripT>"
    is_vulnerable = False
    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:
            pprint(form_details)
            is_vulnerable = True
    return is_vulnerable
```

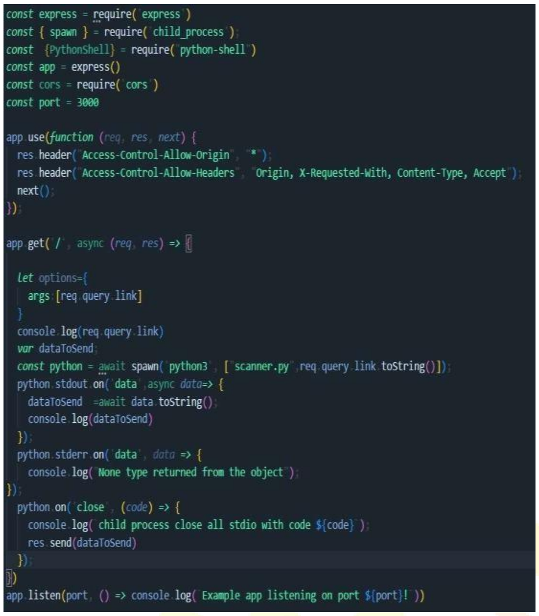
This function runs all above function in sequence



```
if __name__ == "__main__":
    url = sys.argv[1]
    print(str(scan_xss(url)))
```

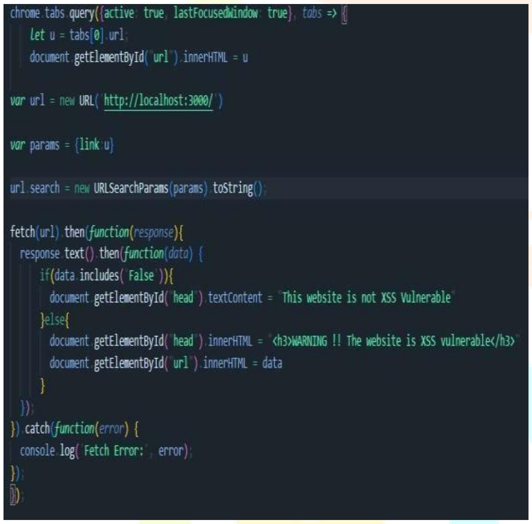
Main Function Accepts website URL as input

- **Server**



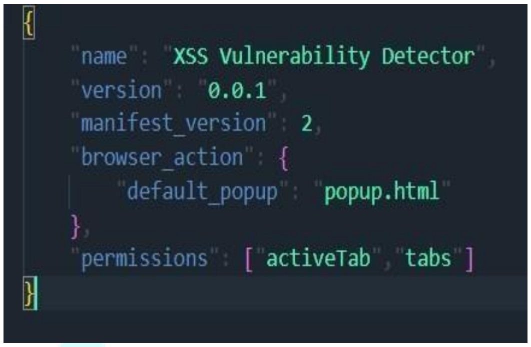
```
const express = require('express')
const { spawn } = require('child_process');
const  {PythonShell} = require("python-shell")
const app = express()
const cors = require('cors')
const port = 3000

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});

app.get('/', async (req, res) => {

  let options={
    args:[req.query.link]
  }
  console.log(req.query.link)
  var dataToSend;
  const python = await spawn('python3', ["scanner.py",req.query.link.toString()]);
  python.stdout.on('data',async data=> {
    dataToSend  =await data.toString();
    console.log(dataToSend)
  });
  python.stderr.on('data', data => {
    console.log("None type returned from the object");
});
  python.on('close', (code) => {
    console.log(`child process close all stdio with code ${code}`);
    res.send(dataToSend)
  });
})
app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

- **Index.js**



```
chrome.tabs.query({active: true, lastFocusedWindow: true}, tabs => {
    let u = tabs[0].url;
    document.getElementById("url").innerHTML = u

var url = new URL('http://localhost:3000/')

var params = {link:u}

url.search = new URLSearchParams(params).toString();

fetch(url).then(function(response){
  response.text().then(function(data) {
      if(data.includes('False')){
        document.getElementById("head").textContent = "This website is not XSS Vulnerable"
      }else{
        document.getElementById("head").innerHTML = "<h3>WARNING !! The website is XSS vulnerable</h3>"
        document.getElementById("url").innerHTML = data
      }
  });
}).catch(function(error) {
  console.log('Fetch Error:', error);
});
});
```

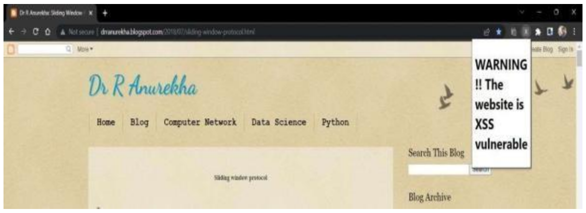
- **Extension of the js file**



```
{
    "name": "XSS Vulnerability Detector",
    "version": "0.0.1",
    "manifest_version": 2,
    "browser_action": {
        "default_popup": "popup.html"
    },
    "permissions": ["activeTab","tabs"]
}
```

RESULTS

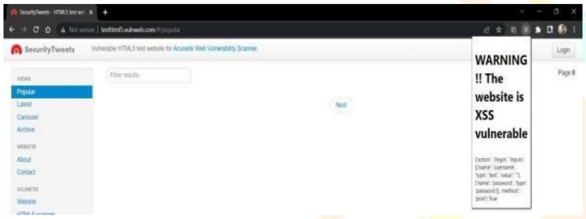
a. Link- http://drranurekha.blogspot.com/2018/07/sliding-window-protocol.html



WARNING !! The website is XSS vulnerable

As displayed by the extension, this webpage is vulnerable to Cross-site Scripting attack

b. Link- http://testhtml5.vulnweb.com/#/popular

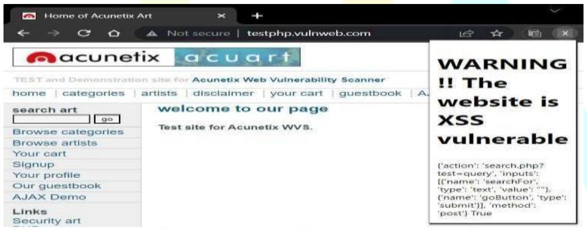


WARNING !! The website is XSS vulnerable

{'action': '/login', 'inputs': [{'name': 'username', 'type': 'text', 'value': ''}, {'name': 'password', 'type': 'password'}], 'method': 'post'} True

As displayed by the extension, this webpage is vulnerable to Cross-site Scripting attack

c. Link - http://testphp.vulnweb.com/

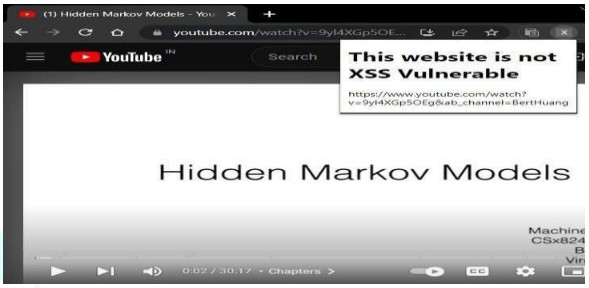


WARNING !! The website is XSS vulnerable

{'action': 'search.php?test=query', 'inputs': [{'name': 'searchFor', 'type': 'text', 'value': ""}, {'name': 'goButton', 'type': 'submit'}], 'method': 'post'} True

As warned by the extension, this webpage is vulnerable to XSS attack

d. Link - https://www.youtube.com/

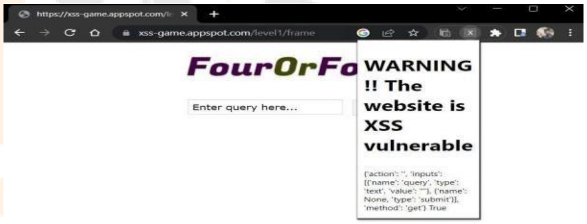


This website is not XSS Vulnerable

https://www.youtube.com/watch?v=9yl4XGp5OEg&ab_channel=BertHuang

YouTube is a secure website and hence is not vulnerable to XSS

e. Link - https://xss-game.appspot.com/level1/frame



WARNING !! The website is XSS vulnerable

{'action': '', 'inputs': [{'name': 'query', 'type': 'text', 'value': ""}, {'name': None, 'type': 'submit'}], 'method': 'get'} True

As warned by the extension, this webpage is XSS vulnerable

REFERENCES

1. GA XSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm – https://www.hindawi.com/journals/scn/2022/2031924/

2. Web Vu lnerability Finder(W VF): Automated Black-Box Web Vulnerab ility Scanner- https://www.mecs-pres.org/ijitcs/ijitcs-v12-n4/v12n4-5.html

3. Detecting Cross-Site Scripting in Web

4. Applications Using Fuzzy Inference System – https://www.hindawi.com/journals/jcnc/2018/8159548/

5. Cross-Site Scripting(XSS) Detection Integrating Evidences in Multiple Stages- https://dblp.org/rec/conf/hicss/hangJL19/html

6. DeepXSS: Cross Site Scripting Detection Based on Deep Learning – https://dl.acm.org/doi/abs/10.1145/3194452.3194469