



Indian Institute of
Technology Bombay

CS 416 M: Computer & Network Security

Course Project

WAVS: Web App Vulnerability Scanner

A Virtual Server to Assess Vulnerabilities & Security Features of Web Apps

Team: The Bug Slayers

Amol Shah	18D070005
Saavi Yadav	170020003
Shreya Laddha	180070054
Tezan Sahu	170100035

Spring 2020-21

Contents

Contents	i
1 Project Overview	1
1.1 Problem Statement	1
1.2 Final Deliverables	2
2 Software System Architecture	3
2.1 Design Requirements:	4
2.2 Libraries/Packages Used	4
3 Security Aspects: <i>A Deep Dive into the Vulnerabilities that WAVS Detects</i>	5
3.1 SSL/TLS Certificates	5
3.2 Cross-Site Scripting (XSS)	6
3.3 Phishing	6
3.4 Open Redirects	7
4 Implementation Details	8
4.1 SSL/TLS Certificate Verification Module	8
4.2 Phishing Website Detection Module	8
4.3 Open Redirect Detection Module	9
4.4 Cross-Site Scripting (XSS) Detection Module	10
4.5 WAVS Virtual Server	10
4.6 Proxy Server	11
5 Critical Evaluation	12
6 Demonstration: <i>Scanning of Websites using WAVS</i>	14
7 Wrapping Up	17
7.1 Individual Contributions	17
7.2 Future Project Suggestions	17
8 Annotated References	18

Project Overview

1.1 Problem Statement

The aim here is to develop **WAVS (Web App Vulnerability Scanner)**, a tool to scan & test URLs for certain vulnerabilities & security issues by simply inspecting the corresponding client-side website. The working of WAVS as an *inline service* is conceptualized in Figure 1.1, which depicts that a user can send a *target URL* to WAVS, which performs the required vulnerability scans & shows a detailed dashboard containing the results.

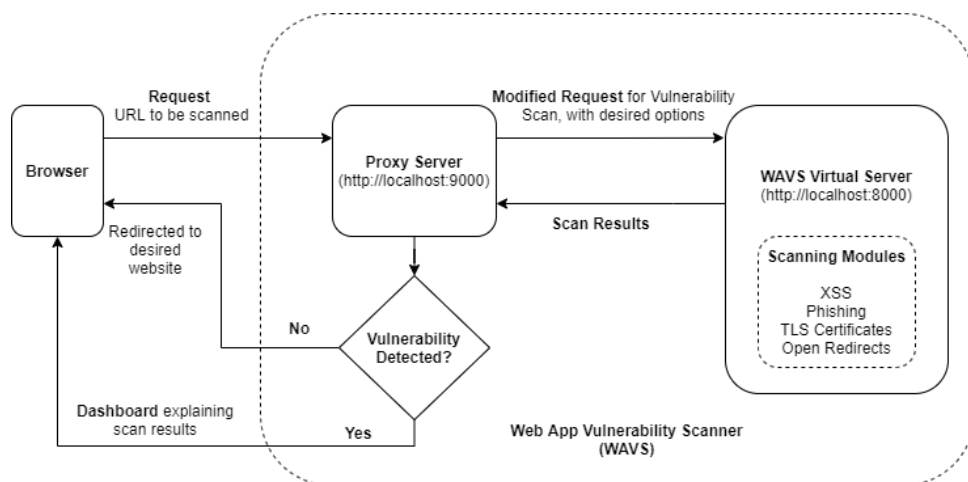


Figure 1.1: Basic Functioning of the Web App Vulnerability Scanner

The internal system of WAVS includes a virtual server with modules for detecting the different vulnerabilities, along with a proxy server, to direct requests from a browser to the virtual server first while visiting a website. The proxy could warn the user before redirecting to the website if some vulnerabilities are found during the scan done by our virtual server.

We intend to identify & assess the following classes of vulnerabilities that a website may possess:

- Absence of Valid TLS Certificates
- Cross-Site Scripting (XSS)
- Potential Phishing Attempts
- Open Redirection

1.2 Final Deliverables

- A **virtual server** capable of identifying the following **client-side security issues**, given URLs:
 - Absence of Valid SSL/TLS Certificates
 - Cross-Site Scripting (XSS)
 - Potential Phishing Attempts
 - Open Redirection
- A **proxy server** to redirect web traffic from a user's browser to our virtual server for scanning the URL for potential threats & return a dashboard containing the scan results
- A **templated dashboard** to display the various details of the different vulnerability scans performed by WAVS as shown in Fig. 1.2b
- A dummy **website with vulnerabilities** introduced deliberately for WAVS to detect them
- A **full-fledged working demo** of WAVS, along with the key learnings from this project
- A **comprehensive report** outlining the architecture and design of WAVS, along with the security aspects & individual contributions



(a) Landing Page of WAVS Proxy

(b) WAVS Dashboard for a URL scan

Figure 1.2: UI of the Web App Vulnerability Scanner (WAVS)

Project Repository

GitHub repository containing the code for our project: [WAVS: Web App Vulnerability Scanner](#)

Software System Architecture

The system architecture of WAVS, with all the necessary components, can be found in Figure 2.1.

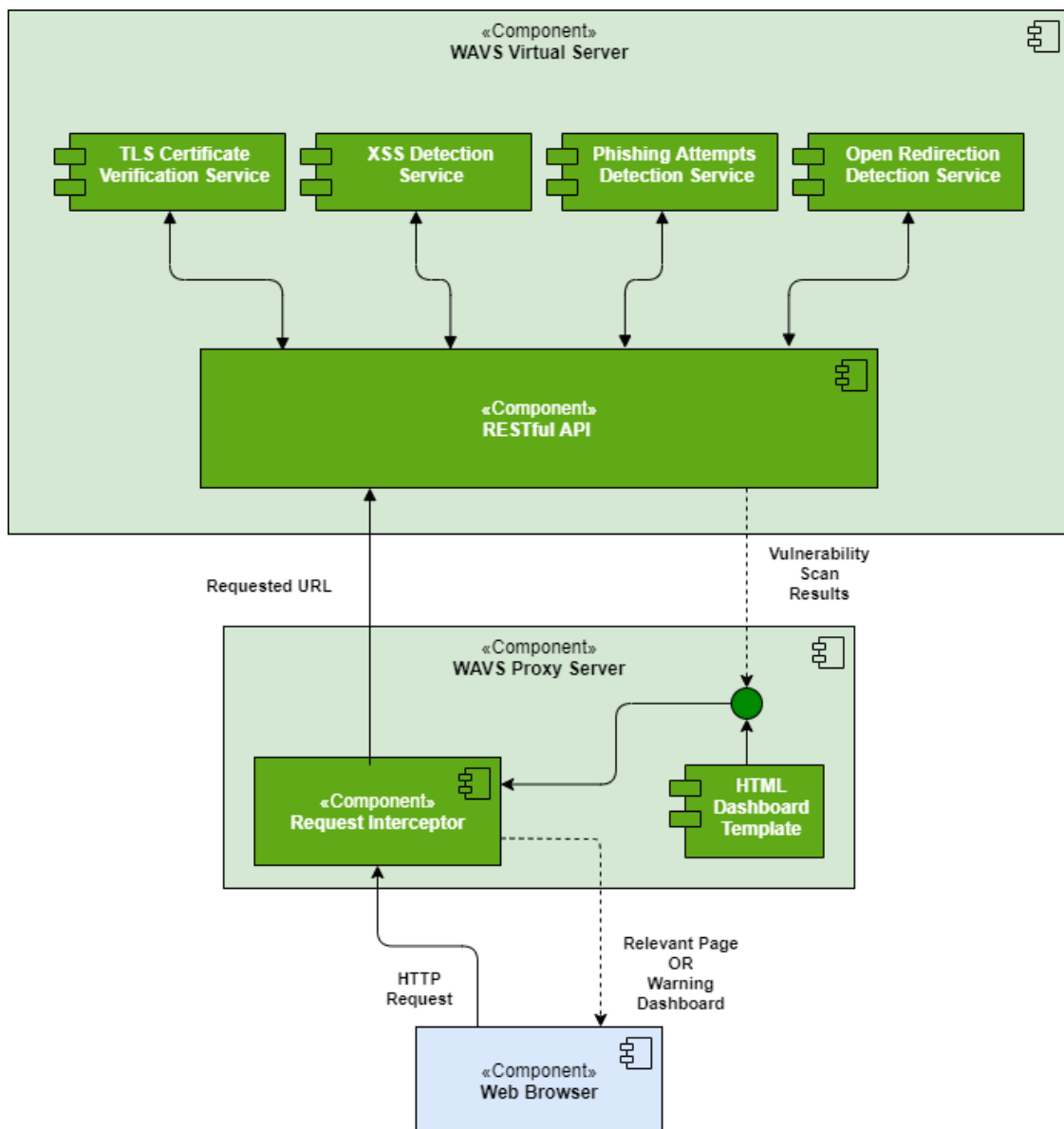


Figure 2.1: Overall System Architecture of WAVS

Each of the vulnerability identification components of the WAVS virtual server has been implemented as a separate microservice & integrated with the virtual RESTful API server. Each of these microservices can be enabled or disabled based on the preferences indicated by the user in the body of the HTTP request. (*default being a **Full Scan***).

2.1 Design Requirements:

Following are some of the design requirements that are critical to the development & functioning of WAVS as an inline tool for performing scans on the various websites requested by the user:

- **Accuracy of Vulnerability Scan:** Results of the scans conducted by each module should be reasonably accurate in order to minimize false alarms, while still warning the users about potential security threats present in a website. To achieve this, we tried to bring together the best of functionalities of some existing libraries for each vulnerability that we address in WAVS.
- **Latency:** The entire scan for a given URL should be conducted such that the users do not have to wait for an extremely long time to browse websites. To address this, we tried to prune down the number of features that need to be extracted inline for phishing detection. Currently, it takes **10-15 seconds** to obtain the results of a full scan for a URL.

Another important approach could be to *cache* the results of frequently requested URLs in order to improve latency, but that is *beyond the scope of this course project*.

- **Modularity & Scalability:** Having the code separated into modules interacting with each other is key for scalability. Hence, we conduct each vulnerability scan using a separate module & combine the results later. This also allows for disabling certain scans based on user requirements.

2.2 Libraries/Packages Used

All the modules, along with the server & proxy have been developed & integrated by our team, after drawing inspiration for various online tools & libraries that address many such security threats individually. Following is a list of major libraries that we leverage to implement the various components of WAVS:

WAVS Component	Language	Major Modules/Packages Involved
TLS Certificate Verification	Python	check-tls-certs , pyOpenSSL
XSS Detection	Python	requests , bs4 , urllib.parse
Phishing Website Detection	Python	tldextract , bs4 , scikit-learn
Open Redirection Detection	Python	requests , urlparser
WAVS Server	Python	FastAPI
Proxy Server	NodeJS	hoxy , ejs , commander
Vulnerable Website	HTML/NodeJS	express

Security Aspects

A Deep Dive into the Vulnerabilities that WAVS Detects

3.1 SSL/TLS Certificates

TLS (*Transport Layer Security*) & SSL, (*Secure Socket Layers*) are both cryptographic protocols that encrypt data and authenticate a connection when moving data on the Internet. TLS is actually just a more recent version of SSL.

Digital certificates, also known as public key certificates, are digital files that prove who owns a public key. A Certificate Authority (CA) issues TLS certificates, which are a form of digital certificate. The CA certifies that the certificate belongs to the owners of the domain name that is the subject of the certificate by digitally signing it. TLS certificates usually contain the following information:

- The subject domain name
- The subject organization
- The name of the issuing CA
- Additional or alternative subject domain names, including subdomains
- Issue date/Expiry date
- The public key
- The digital signature by the CA

Certificate Chain

A certificate chain is a set of certificates (usually beginning with an end-entity certificate) followed by one or more CA certificates (usually ending with a self-signed certificate), with the following properties:

- Each certificate's issuer (except the last) corresponds to the subject of the next certificate
- The secret key corresponding to the next certificate in the chain is expected to sign each certificate (except the last one).
- The *trust anchor* is the last certificate on the list: a certificate that you trust because it was given to you through a reliable process. A trust anchor is a CA certificate that is used as the starting point for path validation by a relying party.

The certificate chain or chain of trust is defined as *certification path*. In order for an TLS certificate to be trusted, it has to be traceable back to the trust root it was signed off, meaning all certificates in the chain—server, intermediate and root—need to be properly trusted.

Vulnerabilities due to Absence of Certificates

- **Unknown, Untrusted, and Forged Certificate Authorities:** Maintaining the trust required for today's global business demands a known and reputable CA that both parties can rely upon to authenticate the conversation.
- **Expired SSL/TLS Certificates:** Expired certificates either cause unplanned system outages or open a door through which hackers can enter your network, or both. An SSL/TLS session that uses an expired certificate should not be trusted. Accepting an expired certificate makes users vulnerable to man-in-the-middle (MITM) attacks.
- **Phishing Scams:** During a scam, when consumers get sent to a website that is not secured by SSL, they are able to notice it. The website's address bar won't have the lock icon or the "https." SSL can't completely prevent phishing, but it makes browsers and consumers more cautious to use only authenticated sites.

3.2 Cross-Site Scripting (XSS)

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

Types of XSS Vulnerabilities

There are three main types of XSS attacks. These are:

- **Reflected XSS** arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.
- **Stored XSS** (also called persistent or second-order XSS) arises when an application receives data from an untrusted source & includes it within its later HTTP responses in an unsafe way.
- **DOM-based XSS** (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

3.3 Phishing

Phishing is a form of fraud in which the attacker tries to learn sensitive information such as login credentials or account information by sending as a reputable entity or person in email or other communication channels. What really distinguishes phishing is the form the message takes: the attackers

masquerade as a trusted entity of some kind, often a real or plausibly real person, or a company the victim might do business with.

Types of Phishing

- **Spear Phishing:** It targets a specific person or enterprise, as opposed to random application users & is a more in-depth version of phishing that requires special knowledge about an organization
- **Whaling:** These target an organization's senior or C-level executives & take specific responsibilities of these executive roles into consideration, using focused messaging to trick the victim

Phishing Techniques

- **Link Spoofing:** One common deception attackers use is making a malicious URL appear similar to an authentic URL, increasing the likelihood that a user will not notice the slight difference(s) and click the malicious URL
- **Website Spoofing:** Websites can be spoofed or forged to appear as if they are the authentic, legitimate site by utilizing things such as Flash or JavaScript, allowing attackers to control how the URL is displayed to the targeted user

3.4 Open Redirects

Open Redirection is when a web application or server uses an unvalidated user-submitted link to redirect the user to a given website or page. Since the domain name in a URL is typically the only indicator for a user to recognize a legitimate website from a non-legitimate one, an attacker can abuse this trust to exploit an open redirect vulnerability on the vulnerable website and redirect the user to a malicious site to execute further attacks.

Types of Open Redirection

- **Header-based:** Header-based being a location-header sent from the server. The benefit with this, for an attacker's perspective, is that the redirect always works even if Javascript is not interpreted. A server side function that gets a URL as input will follow the redirect and end up somewhere else.
- **Javascript-based:** When the redirect instead happens in Javascript, it only works in scenarios where Javascript is actually executed. It might not work for server-side functions, but it will work in the victim's web browser. If the redirect happens in Javascript it might also be possible to cause a redirect to `javascript:something()`, which would be an XSS in itself.
- **Meta-tag based:** An HTML `<meta>` element specifies the time in seconds before the browser is to refresh the page. Providing an alternate URL allows the element to be used as a timed URL redirector. Attackers can use this redirect behavior in situations where they have the ability to control the content attribute of a meta tag or to inject their own tag via some other vulnerability.

Implementation Details

4.1 SSL/TLS Certificate Verification Module

This module has been developed to detect the presence & validity of SSL/TLS certificates for a given URL by obtaining the chain/hierarchy of certificates for the website & validating each of them while checking for their expiry.

- The pyOpenSSL library is used to validate the certificate chain. We are also checking the domain and certificate chain for certificate issuer common name, signature algorithm and expiry date
- The module returns a information about each of the certificate in the chain/hierarchy along with errors/warnings encountered
- Warning & errors are issued for abnormal behaviours detected in the above entries:
 - A warning indicates the impending expiry of one of the certificates in the chain. Presence of warnings only result in an overall ‘Valid (with Warnings)’ status.
 - An error indicates either the absence of a valid certificate or the presence of an insecure signature algorithm on some certificate. These result in an overall ‘Invalid’ status.
- If the entire certificate chain/hierarchy for a website is verified properly, a ‘Valid’ status is returned

4.2 Phishing Website Detection Module

A **traditional ML-based classifier** (using relevant *page keywords*, *address bar*-based, *HTML/JS*-based & *domain*-based features from the website) has been developed as the heart of this module (inspired from the technique mentioned in [this paper](#)). When a URL is sent for scanning, all the required features for the webpage are extracted & run through our classifier to identify the website as *phishing* or not.

- The [UCI Phishing Websites dataset](#), with 30 features was used to train several classifier models to detect potential phishing attempts on a website given its URL.
- The **Gradient Boosting (GB) Classifier**, which gave the highest accuracy of **97.06 %**, was chosen to be a part of this module.
- To streamline the feature-extraction process (given any URL), we pruned the feature set for the GB classifier above to use only the **top 20 most significant features** for phishing detection such that the accuracy did not drop by much. Table 4.1 shows the details of these 20 features that are extracted for every URL to perform the classification.

Feature Name	Description
SSLfinal_State	Checks the age & issuer of SSL Certificate for website
URL_of_Anchor	% of URLs in pointing to different domains or not to any webpage
Links_in_tags	% of links in <script>, <link> & <meta> tags with different domains
web_traffic	Popularity of a website using ranks from the Alexa database
Prefix_Suffix	Prefixes or suffixes separated by (-) to the domain name
having_Sub_Domain	Existence of multiple subdomains in the URL
SFH	Check if the Server Form Handler has about:blank
Request_URL	% of external objects within a webpage loaded from different domain
Links_pointing_to_page	Number of backlinks pointing to the page
Google_Index	Check if the page is in Google's index or not
URL_Length	Length of the URL (longish URLs are considered phishy)
DNSRecord	Existence of a DNS record for the webpage in the WHOIS database
Domain_registration_length	Registration period of domain & time until expiration
having_IP_Address	Presence of an IP address (decimal/hex) in the domain part of URL
HTTPS_token	Existence of <i>HTTPS</i> Token in the Domain Part of the URL
Page_Rank	Google's PageRank value for a webpage
age_of_domain	Time since creation of domain name of the website
popUpWindow	Existence of popups such as prompt() or alert() in the webpage
Iframe	Presence of <iframe> in a webpage to display additional webpages
onmouseover	Use of onmouseover() event to change address bar contents

Table 4.1: Description of the top 20 features used by our classifier to detect if a website is phishing or not

- A **feature extractor** (extractor.py) was developed to obtain the 20 selected features. Given a URL, the extractor can return all the 2 features in ~ 2-5 seconds.
- The detector.py contains the actual module, which leverages the feature extractor & the GB classifier, & would be integrated into the virtual server. Given a URL, it first completes some obvious prechecks, then extracts the features for the website & passes this feature vector into the classifier for prediction. Appropriate results are returned in a JSON format.

4.3 Open Redirect Detection Module

This module has been designed to detect header based open redirects vulnerabilities on websites by fuzzing the URL provided as the input.

- The redirection is made based on a parameter value set through the URL. Table 4.2 gives a list of such parameters. In our module, the URL is parsed using urlparser library to obtain the parameter and its value to get the redirect-to url in case of a open redirect detection.

url	rurl	u	next
link	lnk	go	target
dest	destination	redir	redirect_uri
redirect_url	redirect	r	view
loginto	image_url	return	returnTo
return_to	continue	return_path	path

Table 4.2: URL parameters used to handle redirection

- A redirection payload, namely `https://www.google.co.in`, is injected into the URL and a HTTP request is sent using the `requests` library. If the fetched HTTP status code is a redirect code (3xx) then it classifies as 'Header Based Redirection'.
- The module also verifies whether the URL to which the webpage is getting redirected is same as the payload injected or the redirect-to URL obtained from a parameter

4.4 Cross-Site Scripting (XSS) Detection Module

This module has been implemented by scanning the HTML code of the webpage for input sources and injecting payloads in it to detect all three types- Reflected, Stored and DOM XSS vulnerabilities.

- To implement this, we first extracted the HTML code of the webpage using the `BeautifulSoup` library and searched if for input tags under the form tag. This covers search bars, input textboxes and forms present on the webpage.
- After determining the possible injection points, we extract more details about the injection points such as the 'action' (specifies where to send the form-data when a form is submitted) and 'method' (specifies the HTTP method to use when sending form-data) attributes
- Post that from a payload file, for each payload we inject it by sending a GET or POST request using the `requests` library.
- Then we analyse the content of the response and if the payload was successfully injected then we return the details of the state of vulnerability of the injection point if vulnerability found.

The following features are implemented:

- **Blind XSS Support:** Blind XSS occurs when the attacker input is saved by the web server and executed as a malicious script in another part of the application or in another application.
- **Automatic Payload Generation:** Payloads are sent as input into predetermined sections of the website to detect XSS vulnerabilities

4.5 WAVS Virtual Server

This is the WAVS RESTful API server that can accept requests for scans, leverage the above modules to complete the required scans & return the results as a JSON object.

- FastAPI has been used to set up the server & is made to run locally on port 9000
- All the 4 vulnerability detection modules have been integrated into the server to allow for the corresponding scans
- Requests for scans can be made to the `/v1/scan/` endpoint with the `url` parameter containing the URL of the website to be scanned along with a request body indicating the `ScanOptions`
- The `ScanOptions` mentioned above can be used to indicate which scans are to be performed for a website (by default, all scans will be performed)
- The CORS Middleware of FastAPI allows only the requests from our proxy (running on port 8000) to hit the WAVS server

Note: 'Cross-origin resource sharing (CORS)' is a mechanism that allows restricted resources on a web page to be requested from a domain outside the domain from which the first resource was served

4.6 Proxy Server

The WAVS proxy has been created to intercept requests made from web browsers (by the user) & send the requested URL to our virtual server to test for potential vulnerabilities. On receiving a response from the WAVS virtual server, it returns the results to the user in the form of a dashboard created by injecting the relevant information about the detected vulnerabilities into an HTML template.

- The interceptor for our proxy server has been implemented using hoxy, an open source HTTP hacking API for Node.js. It functions as a normal proxy standing between the client (browser) and server, where one can intercept traffic during both the request and response phase.
- The proxy is made to start on `http://localhost:8000/`, but can be made to run on a different port and host as well.
- Target URL is fetched from the request provided like `http://localhost:8000/<target>` & is sent to the `/v1/scan/` endpoint of the virtual server with appropriate request parameter & body
- A dashboard, templated using ejs (a simple templating language for generating HTML markup with plain JavaScript) is used to display the scan results obtained as a response from the WAVS server to the user
- A command line interface is developed using commander for starting the proxy server, allowing the user to choose vulnerabilities to be scanned by passing corresponding parameters as arguments. This can be seen in Fig. 4.1.

```
tezansahu@TEZANSAHU:/mnt/d/Tezan/Acads/Sem8/IITB_Courses/CS416M/Project/WAVS/proxy$ node wavs_proxy.js --help
Usage: wavs_proxy [options]

Options:
  -s, --scan <type>      Scan Type ("full" or "selective") (default: "full")
  -t, --tls_cert           Enable scan for SSL/TLS Certificates
  -x, --xss               Enable scan for Cross-Site Scripting
  -p, --phishing          Enable scan for potential Phishing
  -o, --open_redirect     Enable scan for Open Redirects
  -h, --help              display help for command
tezansahu@TEZANSAHU:/mnt/d/Tezan/Acads/Sem8/IITB_Courses/CS416M/Project/WAVS/proxy$
```

Figure 4.1: Options that can be supplied to the WAVS proxy at its start to perform full or selective scans

Critical Evaluation

In this section, we try to explain some of our choices made while developing the modules for WAVS along with the challenges faced & critical appreciation.

Phishing Website Detection Module:

The objective was to come up with a reasonable classifier that could distinguish between potential phishing, suspicious & legitimate websites using features extracted from a URL. For this, we tried to train several types of classifiers using the UCI Phishing Websites dataset, compared their accuracies & chose the best model. A summary of the results obtained can be found below. Details of all the experiments can be found in the `PhishingDetectionExpt.ipynb` notebook.

Classifier Model	Accuracy (%)
SVM (Linear Kernel)	92.85
SVM (RBF Kernel)	96.38
k-NN ($k = 1$)	95.43
Decision Tree (with gini)	96.02
Decision Tree (with entropy)	96.79
Random Forest (Estimators = 100, Depth = 18)	96.61
Gradient Boosting (Estimators = 100, Depth = 5)	97.06

From the above table, it is evident that the Gradient Boosting classifier outperformed the others & hence, has been chosen to be used in the Phishing Website Detection module of WAVS.

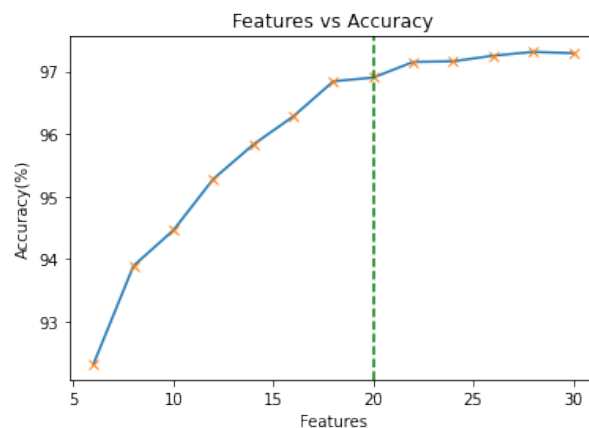


Figure 5.1: Number of features (N) vs Classifier Accuracy for the Feature-Ablation Analysis

We do realize that greater number of features aid in better classification, but we also need to note that since all these features need to be extracted at runtime for a given URL, we need to minimize the time taken to arrive at a prediction without compromising the accuracy by much. Thus, we now perform a **feature-ablation analysis** wherein we retrain the GB classifier using the **Top N most significant features** out of our 30 features in our dataset in order to select a minimum subset of features that would allow our model to classify websites with a considerable accuracy without consuming a long time.

From Fig. 5.1, we observe that we obtain an *elbow point* when we select around 20 features among all the available features. This marks a suitable tradeoff between the accuracy & the latency that we wish to achieve. Hence, we use the 20 most significant features for our model prediction.

TLS Certificate Verification Service:

The Chain of Trust refers to the domains' SSL certificate and how it is linked back to a trusted Certificate Authority. In order for an SSL certificate to be trusted it has to be traceable back to the trust root it was signed off of, meaning all certificates in the chain – server, intermediate, and root, need to be properly trusted. A single root certificate can be used to validate the site on a primary level. If the certificate is valid and can be chained back to a trusted root, it will be trusted. For these reasons, we check the chain instead of the root certificate only.

Open Redirection

For this project, we focus only on the header-based open redirection, which is the easiest and the most common form of open redirection. Dealing with <meta>-tag based and JavaScript-based open redirect vulnerabilities have been left for future work.

Usually, a URL of type `abc.com` is redirected to `www.abc.com` by load balancers with a redirect code 301. To avoid detecting such cases as a vulnerability, we verify that URL to which it is redirected to is the same as the payload URL injected.

Cross-Site Scripting (XSS) Detection

Currently the XSS scanner detects all three kinds of XSS vulnerabilities (Reflected, Stored, DOM-based XSS). It works with static pages whose HTML code is readily available. However, when the page is dynamically loaded, then a considerable amount extra effort (like setting up a virtual browser with Selenium) is required to access the HTML code after it is loaded. Similar extended effort would be needed to integrate it with the virtual server and proxy server. Hence, dealing with dynamically generated webpages is beyond the scope of this project & has been left for future work.

Demonstration

Scanning of Websites using WAVS

A (Dummy) Vulnerable Website

To demonstrate the full capacity of WAVS at once, we created a dummy website shown in Fig. 6.1, with some inherent vulnerabilities that WAVS can detect. The website was developed using express (Node.js) & deployed on Heroku. It can be accessed at: <https://bugslayers-cs416-open-redirect.herokuapp.com/>

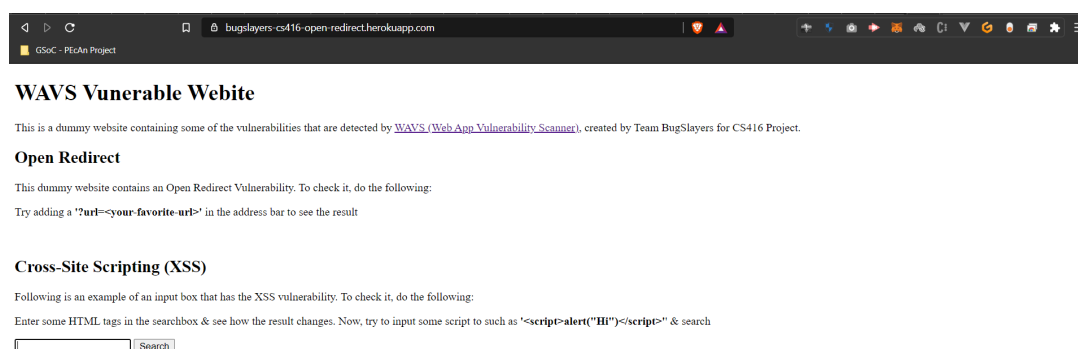


Figure 6.1: Dummy vulnerable website created as a part of this project

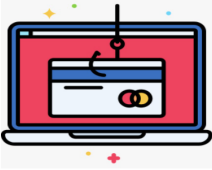
- **Open Redirect** vulnerability was introduced by allowing redirects to any URL if mentioned as the ?url= query parameter in the address
- **Cross-Site Scripting** has been introduced through a small vulnerable form, which allows JavaScript code input & executes it on the browser
- Since the website is recently deployed, & has very few backlinks, ranks low on PageRank & Alexa, it may be considered **potentially phishy** as well
- The certificate chain has some issue with the signature algorithm of one of the SSL certificates, hence the **SSL certificate** verification also fails

Results of Performing a Scan on the Vulnerable Website

The dashboard obtained on performing the vulnerability scan on our dummy website is shown in Fig. 6.2. Details of the individual scans (XSS & SSL/TLS Certificates) are shown in Fig. 6.3 & 6.4.

WAVS Dashboard: Scan Results

URL: <https://bugslayers-cs416-open-redirect.herokuapp.com/>



Potential Phishing Attempts

❌ **Phishing:** This website seems to be *phishy*! We recommend not to visit it

[Click here for Detailed Analysis](#)


Quick Links: [Introduction to Phishing Attacks](#) | [How to Spot a Phishing Website](#)

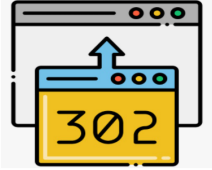
SSL/TLS Certificate Validation

❌ **Invalid:** This website does not have valid SSL/TLS certificate

[Click here for Certificate Details](#)

Quick Links: [Working of SSL/TLS Certificates](#) | [Certificate Chains](#)





Open Redirects

❌ **Vulnerability Detected:** This website has been identified with *Header Based Redirection*

Query Parameter for Redirection: url


Quick Links: [Open Redirect \(Reflected\)](#) | [Impact of Open Redirects](#)

Cross-Site Scripting (XSS)

❌ **XSS Detected:** *DOM-based XSS* vulnerability has been detected in this website

[Click here for Details](#)

Quick Links: [Types of XSS](#) | [XSS Prevention](#)



[Proceed to URL](#)

Copyright ©2021 | Created with ❤️ by Tezan Sahu, Shreya Laddha, Amol Shah & Saavi Yadav

Figure 6.2: WAVS Dashboard after scanning <https://bugslayers-cs416-open-redirect.herokuapp.com/>

[Click here for Certificate Details](#)

XSS Details

Details of the Forms detected with XSS Vulnerability:

Element Name	Element Type	Value Injected
query	text	<script>alert("XSS Vulnerable")</script>

Form Action: /search
Form Submission Method: get

Cross-Site Scripting (XSS)

❌ **XSS Detected:** *DOM-based XSS* vulnerability has been detected in this website

[Click here for Details](#)




Figure 6.3: Details about the XSS Vulnerability

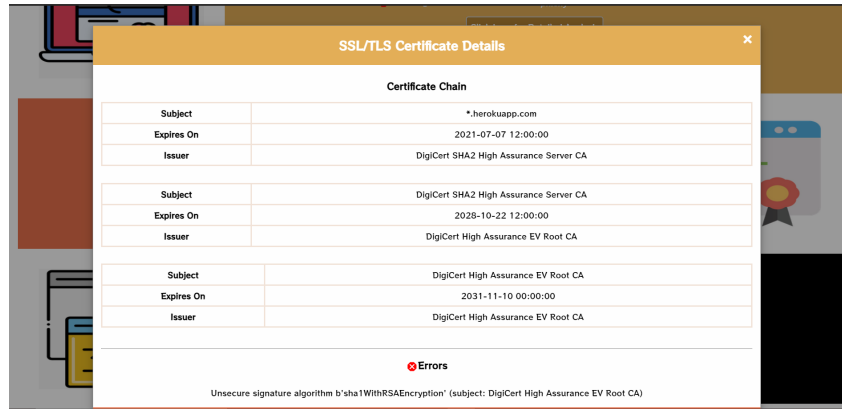


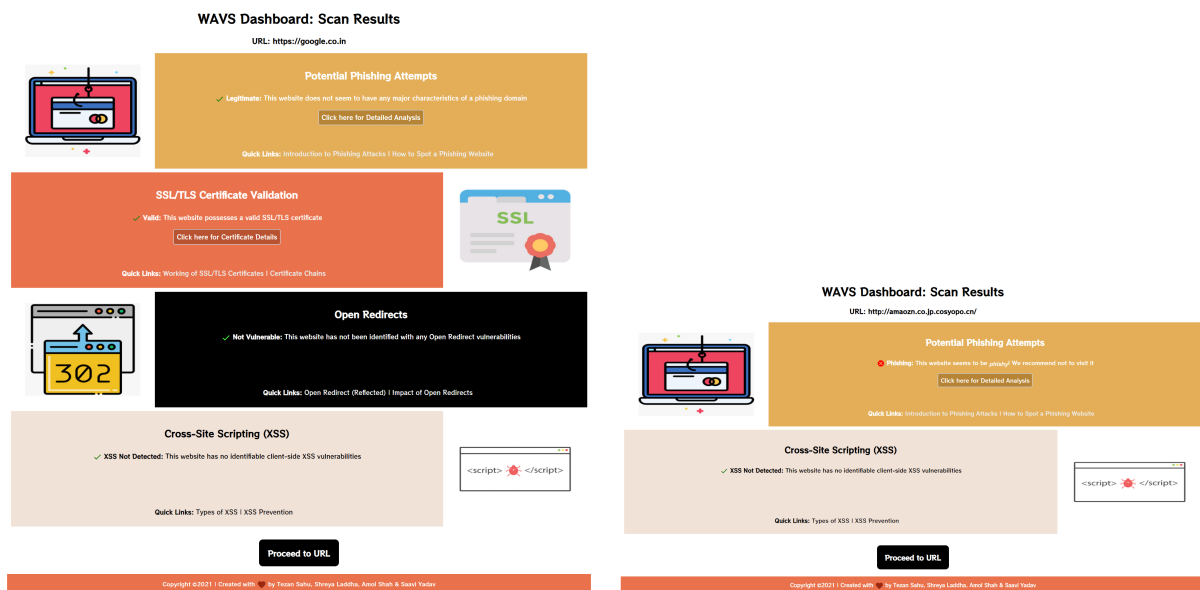
Figure 6.4: Details about the SSL/TLS Certificate Chain

Scanning other Websites

Fig. 6.5a demonstrates the results of scanning the URL `https://google.co.in`. As expected, this website is free from all the four vulnerabilities that WAVS checks for.

Selective Scans

As mentioned previously, the WAVS proxy can be started with flags to allow selective scanning. Fig. 6.5b shows the result of one such scan performed on `http://amaozn.co.jp.cosyopo.cn/` with only the *phishing website* & *XSS detection* modules enabled.



(a) WAVS Dashboard for `https://google.co.in`

(b) Dashboard for Selective Scanning

Figure 6.5: WAVS dashboards for other scans

Wrapping Up

7.1 Individual Contributions

Team Member	Contribution
Amol	XSS (research + module), Reports
Saavi	TLS Certificates (research + module), Reports
Shreya	Open Redirects (research + module), UI, Reports
Tezan	Phishing (research + module), Virtual Server, Proxy, Vulnerable Website, Reports

Table 7.1: Overall contribution of teammates in developing the various components of WAVS

7.2 Future Project Suggestions

Another vulnerability possible to detect is cross-site request forgery (CSRF). More information about it can be found [here](#). It is slightly more complicated & a good learning opportunity which will be a nice addition to the bouquet of vulnerabilities being detected. Other vulnerabilities that can be considered to be added are directory traversal, command injection & remote file inclusion.

Note for Forthcoming Batches

- **Advice on Topic Selection:** OWASP should be a good starting website to get some ideas and information too. Past projects done in the scope of college courses (in or outside of IITB) can also be a good starting point. Take care not to select a too ambitious initial target or a too liberal one - you can consult your mentors regarding this. Keep in mind that the learning something new should be the core objective.
- **Time Management and Teamwork:** Try to work on the project when you are relatively free as you go along the semester and not leave it till the last minute for smooth progress. Be sure to keep communicating with your team members for smooth functioning. Keep timely reminders about the work to be done. It is also a good idea to have one person taking the lead who can initiate discussions and keep the team on track.
- **Learning Strategy:** Firstly one should know the basics of the topic at hand by researching it through the net/books etc. Then you should play around with the existing tools, if any, to get the feel of what you are doing. For example, if you are detecting XSS vulnerabilities, you can try [Google XSS Game](#) out. Not only do you learn about the topic but also have fun along the way.

Annotated References

Following is an annotated list of references that we provide to understand the various theoretical & practical aspects of our project. For each vulnerability that we are dealing with in WAVS, we provide a set of resources to dig deeper into the manifestations & causes of the vulnerability, some techniques to detect or prevent them, as well as some resources that aided us in implementing the code for WAVS.

TLS Certificate Verification

- [What is a TLS/SSL certificate, and how does it work?](#) This blog introduces the concepts of TLS/SSL certificates & their use in maintaining the authenticity & integrity of content on the internet.
- [How Do Certificate Chains Work?](#) This blog presents a deep dive into certificate chains, their working & verification. This forms the basis of the theoretical understanding required to develop the TLS Certificate Verification module.
- [DigiCert SSL Certificate Checker](#) & [SSL Labs Server Test](#) These are online diagnostic tools, which verify the TLS certificate of a given domain. They have been used to understand the outputs that our TLS certificate verification module should automatically present after analysing a URL.

Cross-Site Scripting (XSS) Detection

- [Cross-site Scripting \(XSS\) Tutorial](#) This tutorial introduces the concept of XSS, while illustrating examples for the various types of XSS attacks that are possible.
- [DOM-based XSS](#) This article provides a brief overview about DOM-based XSS attacks, along with some techniques to test & exploit such vulnerabilities. It forms the theoretical basis required for detecting such vulnerabilities automatically using a module.
- [OWASP XSS Filter Evasion Cheat Sheet](#) This article is focused on providing a guide to assist in Cross Site Scripting testing. It provides detailed information about various payloads that can be used for such testing.
- [Test Your XSS Skills Using Vulnerable Sites](#) This website provides links to various XSS playgrounds to tinker around and learn basic XSS attacks

Phishing Detection

- [What is phishing? How this cyber attack works and how to prevent it](#). This article introduces the topic of *phishing* & elucidates the various types & techniques used for such attacks, along with some ways to prevent phishing.
- [How to Spot a Phishing Website](#) This article describes some general ways for users to identify potential phishing websites while surfing on the web
- [Phishing URL Detection with ML](#) This article dives into the use of Machine Learning techniques to identify phishing websites using various features, given a URL
- [Intelligent Rule based Phishing Websites Classification](#) This paper assesses how rule-based classification data mining techniques are applicable in predicting phishing websites, which forms the conceptual bedrock of our implementation of the Phishing Website Detection Module.
- [PhishTank](#) This website offers a community-based phish verification system where users submit suspected phishes and other users *vote* on them. We use URLs from this to test our phishing detection module.

Open Redirect Detection

- [The real impact of an Open Redirect vulnerability](#) This article provides a gentle introduction to Open Redirects, its types & explains various ways in which such a vulnerability can be exploited.
- [URL Redirector Abuse](#) This discussion thread provides a brief overview about the abuse of URL redirection & mentions several ways to implement such redirectors.
- [Oralyzer](#) This tool is a simple python script, capable of identifying the open redirection vulnerability in a website by fuzzing the URL provided as the input. This forms a major inspiration for the implementation of our Open Redirect Detection Module.
- [ORtester](#) This is another tool designed to detect open redirects vulnerabilities on websites through a scan supported by a list of payloads. We again draw some inspiration from this while developing our Open Redirect Detection Module

Miscellaneous

- [Common Web Application Vulnerabilities](#) & [OWASP Top Ten](#) These websites provide a list of some of the most prevalent & notorious web app vulnerabilities, which was leveraged in order to select the vulnerabilities that WAVS could detect & can also be used to extend the capabilities of WAVS in future.
- [Cross-site request forgery](#) This Wikipedia page talks about the basics of CSRF and how it can be prevented.
- [Proxy Server Basics](#) This website highlights the importance of using a proxy, along with the different types and the risks of using a proxy.