Team Name
Bug Slayers

Project Stage II Report

CS416M: Foundations of Network Security and Cryptography

Due Date
20/03/21

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

WAVS: Web App Vulnerability Scanner

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

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, 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.

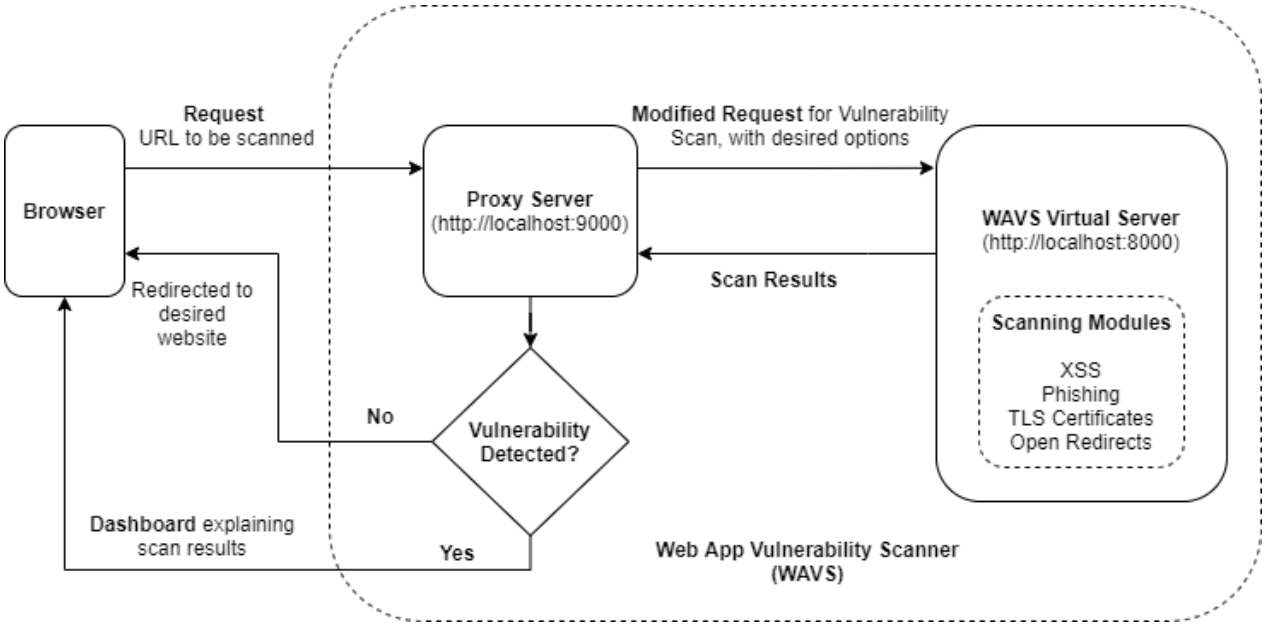

Figure 1: Basic Functioning of the Web App Vulnerability Scanner

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

Project Repository

Link to the GitHub repository containing the code for out project: WAVS: Web App Vulnerability Scanner

Connection with the Course Content

- This project lies majorly in the domain of Application Security, which is an integral part of the course
- By building WAVS, we try to explore the various web app vulnerabilities and create modules to identify them to protect the user
- We also intend to gain some practical experience with current network security tools like Metasploit along with theoretical background

System Software Architecture

The proposed system architecture of WAVS, with all the necessary components, can be found in Figure 2. The **color coding** of the various components depicts the **completion status** of each component: **Green** indicates *fully developed*, **yellow** indicates *work in progress*, while **red** indicates *work not started*.

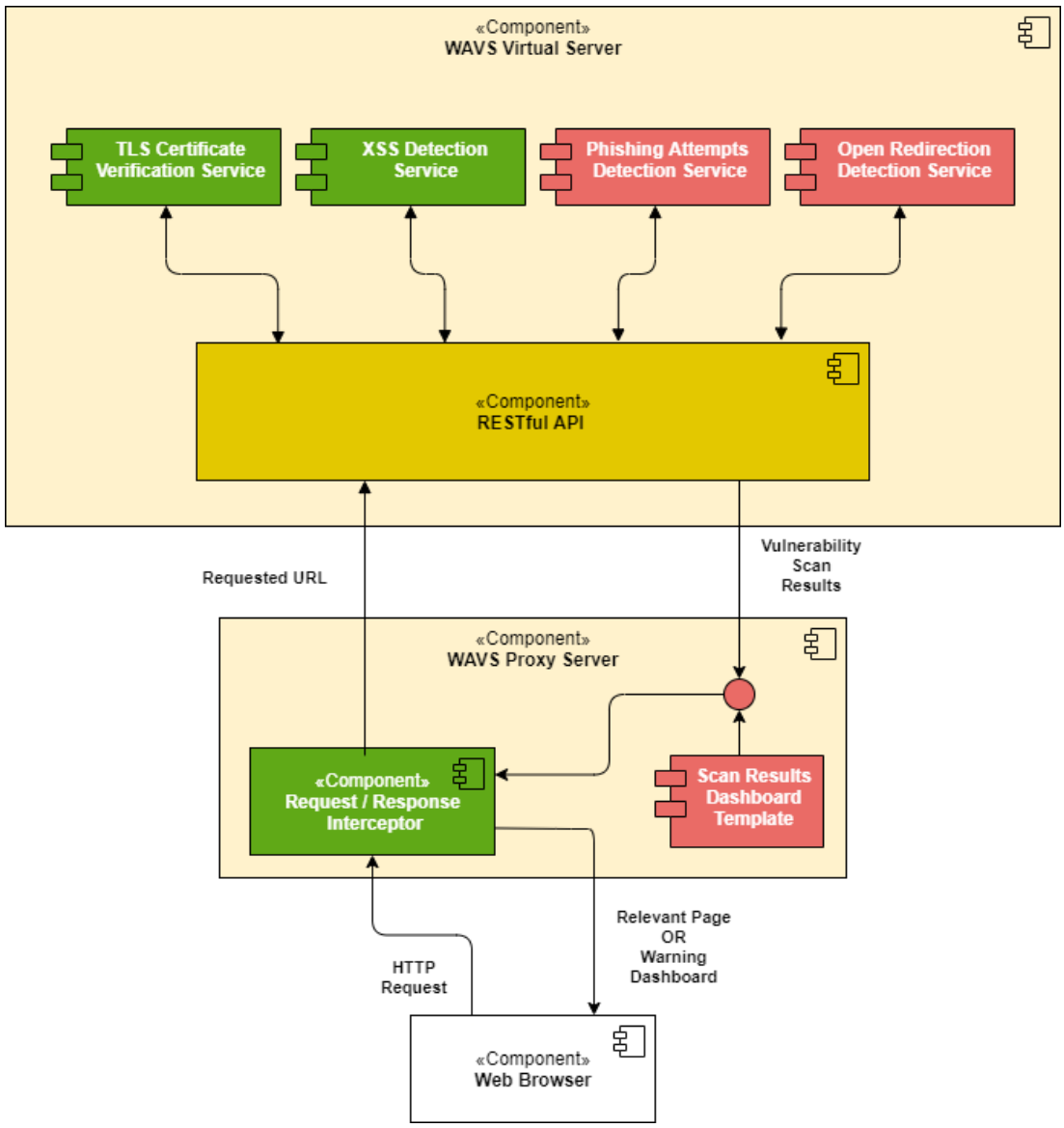

Figure 2: Overall System Architecture of WAVS

Proposed Implementation

Virtual Server

We plan to implement each of the vulnerability identification components of WAVS virtual server as a separate microservice & integrate them 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 HTTP request. *(default being a **Full Scan**)*.

We suggest the following approaches to be taken for identifying each of the different security issues:

- **Absence of Valid TLS Certificates:** To detect the presence & validity of TLS certificates for a given URL, we propose to use necessary libraries to obtain the chain/hierarchy of certificates for the website & validate each of them while checking for their expiry.
- **Cross-Site Scripting (XSS):** We propose to test XSS vulnerabilities by calling various functions to inject XSS payloads into DOM elements like forms & detecting whether the webpage is vulnerable or not.
- **Potential Phishing Attempts:** We intend to create a lightweight **rule-based** (or traditional ML-based) **classifier** using relevant *page keywords*, *address bar*-based, *HTML/JS*-based & *domain*-based features from the website (inspired from the technique developed in this paper). Later, when a URL is sent for scanning, all the required features for the webpage will be extracted & run through our classifier to identify the website as *phishing* or not.
- **Open Redirection:** To address open redirection issues in websites, we propose to work on the lines similar to Oralyzer & ORtester. This involves infusing predefined payload URLs into the given website URL & trying to figure out *Header-based*, *JS-based* & *Meta tag-based* open redirect vulnerabilities.

Proxy Server

The proxy would be created to intercept requests made from web browsers (by the user), test them for potential vulnerabilities by sending the URL to our virtual server, redirect the requests if the URL is secure & warn the user accordingly in case some security issues are detected. The warning will be issued in the form of a dashboard created by injecting the relevant information about the detected vulnerabilities (from the virtual server) into an HTML template, which will be served to the user.

Libraries/Packages To Be Used

All the modules, along with the server & proxy are being 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 Service	Python	`check-tls-certs`, `OpenSSL`
XSS Detection Service	Python	`fuzzywuzzy`, `BeautifulSoup`, `splinter`
Phishing Website Detection Service	Python	`tldextract`, `BeautifulSoup`, `scikit-learn`
Open Redirection Detection Service	Python	`requests`, `BeautifulSoup`
Proxy Server	NodeJS	`hoxy`, `ejs`, `commander`

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, and also plan parallelize the running of each vulnerability scanning module.

  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.

Progress Report

Completed Work

Phishing Website Detection Module:

- 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 integrated into the 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. Given below are 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_registeration_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 *HTTPS* 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
on_mouseover	Use of onmouseover() event to change address bar contents

- 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 $\sim$ 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.

TLS Certificate Verification Service:

- Module for comprehending TLS certificate chains and checking abnormalities in certificates is completed.
- **OpenSSL** 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 dict containing completion status along with errors/warnings encountered.
- Warning and errors are issued for abnormal behaviours detected in the above entries. Warning indicates the incoming expiry of the certificate or unmatched alternate domain names mentioned in the certificates.
- A failed status indicates the presence of one or more errors. In the case of errors, the domain has failed our checks and is not indicated safe for browsing. It shows either the absence of a valid certificate or the presence of an insecure signature algorithm.

Work In Progress

Virtual Server:

Having developed some of the modules by now, we have also set up a basic RESTful API server for WAVS to accept requests for scans, leverage the modules to complete the required scans & return the results as a JSON object. Full integration & testing remains to be done until the other modules are fully developed.

Completed

- We used `FastAPI` to set up the server. By default, it starts on `http://localhost:8000`, but can be made to run on a different host & port as well
- We have integrated the *Phishing Website Detection* & *TLS Certificate Verification* modules 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)

To be Completed

- Integration of the remaining modules after their completion along with formulating a reasonable structure of the JSON object to be returned as response
- Currently the CORS Middleware allows requests from all hosts (`*`). We would later restrict this to only requests coming from the proxy only.

Proxy Server:

We have also set up a Proxy Server that would intercept requests made by the client, send the URL for testing to the virtual server and display the security issues detected on a dashboard.

Completed

- A basic setup for the interceptor of the proxy server has been completed using `hoxy`. `Hoxy` is a completely free, open source HTTP hacking API for Node.js. It functions as a normal proxy standing between the client and server, where we can intercept traffic during both the request and response phase.
- By default, it starts 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>` and is sent to the `/v1/scan/` endpoint of the virtual server with appropriate request parameter & body
- A dummy template is displayed for now as a response from the virtual server using `EJS` which is a simple templating language for generating HTML markup with plain JavaScript.

To be Completed

- A proper template is required to be made for the final dashboard showing the detected security issues. This task will be done once all the other modules are ready as it will require the knowledge of responses of all of them.
- A command line interface is needed to be set up for starting the proxy server, allowing the user to set what all vulnerabilities to be checked by passing corresponding parameters as arguments. This will be done using `commander`.
- Final integration and testing of proxy with virtual server will be the last task.

Future Work

Open Redirection Module:

This module will be designed to detect open redirects vulnerabilities on websites by fuzzing the url provided as the input. We will be focusing on three kinds of vulnerabilities:

- **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.

Cross Site Scripting Module:

This module will be implemented by scanning the HTML code for forms and injecting payloads in it to detect XSS vulnerabilities. The following features will be implemented:

- **Reflected and DOM XSS Scanning:** Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data.

  DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser.
- **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
- **Brute Force Payloads from a File:** Custom payload inputs also can be provided in the form of a file

Contribution of Teammates

The following table summarizes the contribution of the various teammates in developing the various components of WAVS until now:

Name	Individual Contribution
Amol Shah	Researched & explained XSS Detection Fundamentals
Saavi Yadav	Researched & developed TLS Certificate Validation Module
Shreya Laddha	Researched & explained Open Redirection Detection Module
Tezan Sahu	Developed Phishing Website Detection Module, set up Virtual Server & Proxy interceptor

Note: *The slightly disproportionate amount of individual contributions will be suitably accounted for in the next leg of development of the project.*

Critical Evaluation

In this section, we try to explain some of our choices made while developing the modules for WAVS along with some implementation details, 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.

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.

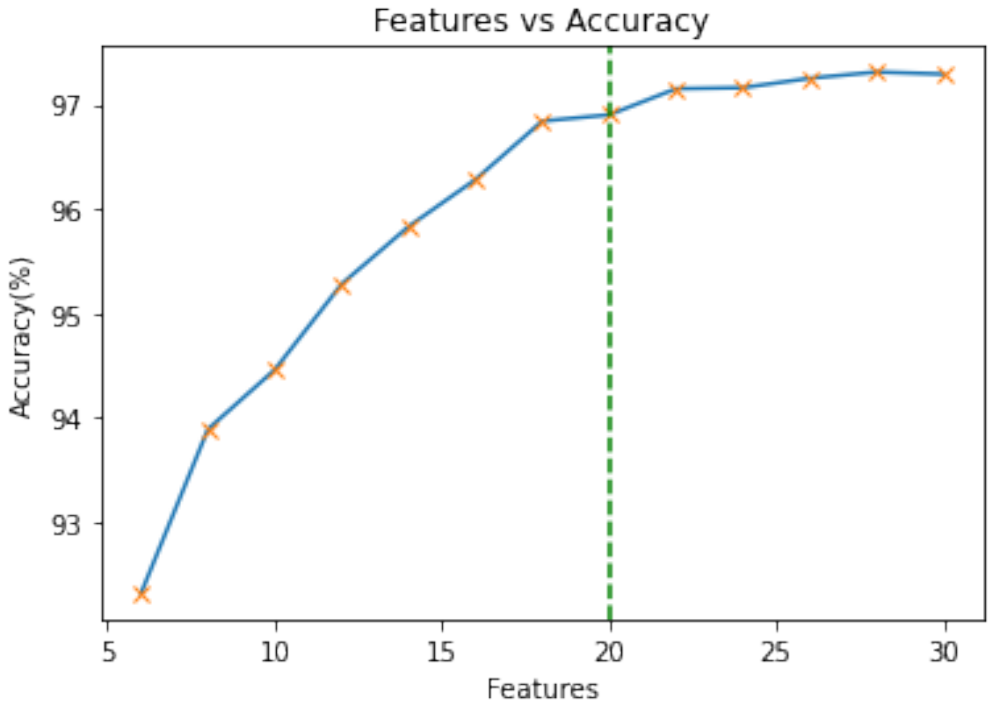

Figure 3: Number of features (N) vs Classifier Accuracy for the Feature-Ablation Analysis

From the plot, 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.

Final Deliverables

- A **virtual server** capable of identifying the following **client-side security issues**, given website URLs:
  - Absence of Valid 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. This would automatically redirect to the website if no vulnerabilities are found. Otherwise, it will present suitable warnings on the user's browser.
- A **full-fledged working demo** of website vulnerability detection using WAVS virtual server & proxy
- **Comparative study** of the functionality & results of WAVS with various existing software having rather similar objectives, some of which include web application firewall (WAF) & SSL labs
- A **presentation** about the nature, sources, impacts & detection techniques of the security issues targeted

Plan of Remaining Work

Time Estimate & Work Allotment

Tasks	Time Estimate	Assignee
XSS Detection Service	3-4 weeks	Amol Shah
Open Redirection Detection Service	2-2.5 weeks	Shreya Laddha
Virtual Server	1 week	Tezan Sahu, Saavi Yadav
Proxy Server	1 week	Shreya Laddha
Comparison with Metasploit	2 weeks	Amol Shah

Effort Estimate

Tasks	Effort Estimate
XSS Detection Service	Develop entire module
Open Redirection Detection Service	Develop entire module
Virtual Server	Integration of 2 modules; JSON response format TBD.
Proxy Server	Server skeleton is ready; dashboard template TBD.
Comparison with Metasploit	Basic familiarization; comparison with our approaches

Future Projects

One more vulnerability possible to implement is cross-site request forgery (CSRF). More information about it can be found here. It is a bit more complicated and a nice 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 and remote file inclusion. One interesting addition, although ambitious, would be adding a part where the vulnerabilities are detected and fixed in the source code.

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 our scanner.

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.

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.

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

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

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.