Server-code

main.py

```
import uvicorn
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi
from starlette.middleware.cors import CORSMiddleware
from fastapi.responses import RedirectResponse
import v1

app = FastAPI()

app.add_middleware(
    CORSMiddleware, allow_origins=["http://localhost:8000"], allow_methods=
["*"], allow_headers=["*"]
)

@app.get("/", tags=["root"])
def get_root():
    return RedirectResponse("/v1/")


app.include_router(
    v1.router,
    prefix="/v1",
    tags=["vulnerability scanning (version 1)"],
    responses={404: {"description": "Not found"}},
)


def custom_openapi():
    if app.openapi_schema:
        return app.openapi_schema
    openapi_schema = get_openapi(
        title="WAVS: Web App Vulnerability Scanner",
        version="0.0.1",
        description="""WAVS (Web App Vulnerability Scanner) is a tool to scan &
test URLs for certain vulnerabilities &
        security issues by simply inspecting the corresponding client-side
website. The overall system would include 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.
        \nWe identify & assess the following security issues that a website may
suffer from: _Absence of Valid TLS Certificates_,
        _Cross-Site Scripting (XSS)_, _Potential Phishing Attempts_ & _Open
```

```
Redirection_
        """,
        routes=app.routes,
    )
    openapi_schema["info"]["x-logo"] = {
        "url": "https://fastapi.tiangolo.com/img/logo-margin/logo-teal.png"
    }
    app.openapi_schema = openapi_schema
    return app.openapi_schema


app.openapi = custom_openapi

if __name__ == "__main__":
    uvicorn.run("main:app", host="localhost", port=9000)import uvicorn
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi
from starlette.middleware.cors import CORSMiddleware
from fastapi.responses import RedirectResponse
import v1

app = FastAPI()

app.add_middleware(
    CORSMiddleware, allow_origins=["http://localhost:8000"], allow_methods=
["*"], allow_headers=["*"]
)

@app.get("/", tags=["root"])
def get_root():
    return RedirectResponse("/v1/")


app.include_router(
    v1.router,
    prefix="/v1",
    tags=["vulnerability scanning (version 1)"],
    responses={404: {"description": "Not found"}},
)


def custom_openapi():
    if app.openapi_schema:
        return app.openapi_schema
    openapi_schema = get_openapi(
        title="WAVS: Web App Vulnerability Scanner",
        version="0.0.1",
        description="""WAVS (Web App Vulnerability Scanner) is a tool to scan &
test URLs for certain vulnerabilities &
        security issues by simply inspecting the corresponding client-side
website. The overall system would include 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.
        \nWe identify & assess the following security issues that a website may
```

```
suffer from: _Absence of Valid TLS Certificates_,
        _Cross-Site Scripting (XSS)_, _Potential Phishing Attempts_ & _Open
Redirection_
        """,
        routes=app.routes,
    )
    openapi_schema["info"]["x-logo"] = {
        "url": "https://fastapi.tiangolo.com/img/logo-margin/logo-teal.png"
    }
    app.openapi_schema = openapi_schema
    return app.openapi_schema


app.openapi = custom_openapi

if __name__ == "__main__":
    uvicorn.run("main:app", host="localhost", port=9000)
```

1. Open Redirection Detection

1.1 detector.py

```
import warnings,ssl,requests
import urllib.parse as urlparse
warnings.filterwarnings('ignore')
ssl._create_default_https_context = ssl._create_unverified_context
#------------------------------------------------------------------------------
----#

class OpenRedirectsDetector:
    def __init__(self):
        self.url = None

    def detect_or(self, url):
        if urlparse.urlparse(url).scheme == '':
            url = 'http://' + url
        self.url = url
        result = self.check()
        return result

    def check(self):
        values = ['url', 'rurl', 'u','next', 'link', 'lnk', 'go', 'target',
'dest', 'destination', 'redir',
        'redirect_uri', 'redirect_url', 'redirect', 'view', 'loginto',
'image_url', 'return', 'returnTo', 'return_to',
        'continue', 'return_path', 'path']
        RedirectCodes = [i for i in range(300,311,1)]

        request = requests.Session()
        parsed = urlparse.urlparse(self.url)
        params = urlparse.parse_qsl(parsed.query)
        if(params):
            try:
                page = request.get(self.url, allow_redirects=False, timeout=10,
verify=False, params='')
                page2 = request.get(self.url, allow_redirects=True, timeout=10,
verify=False, params='')
                # print(page2.request.url)
                if page.status_code in RedirectCodes:
                    details = {}
                    for x,y in params:
                        if(x in values and y==page2.request.url):
                            details = {"parameter": x, "redirect_url": y }
                            break
                    if(details):
                        return {"result": "Header Based Redirection",
"details": details}
                    elif('/r/' in self.url):
                        details = {"parameter": 'r'}
                        return {"result": "Header Based Redirection",
"details": details}
                    else:
                        return {"result": "Open Redirection", "details": {}}
                elif page.status_code==404:
                    return {"result": "404 Not Found", "details": {}}
                elif page.status_code==403:
                    return {"result": "403 Forbidden", "details": {}}
                elif page.status_code==400:
                    return {"result": "400 Bad Request", "details": {}}
            except requests.exceptions.Timeout:
                return {"result": "TimeOut", "details": {"url": self.url}}
            except requests.exceptions.ConnectionError:
                return {"result": "Connection Error", "details": {"url":
self.url}}

        payload = "https://www.google.co.in"
        query = ["url", "redirect_url"]
        for x in query:
            url = self.url+"?"+x+"="+payload
            try:
                page = request.get(url, allow_redirects=False, timeout=10,
verify=False, params='')
                page2 = request.get(url, allow_redirects=True, timeout=10,
verify=False, params='')
                if(page.status_code in RedirectCodes and page2.request.url ==
payload):
                    return {"result": "Header Based Redirection", "details":
{"parameter": x}}
                elif page.status_code==404:
                    return {"result": "404 Not Found", "details": {}}
                elif page.status_code==403:
                    return {"result": "403 Forbidden", "details": {}}
                elif page.status_code==400:
                    return {"result": "400 Bad Request", "details": {}}
            except requests.exceptions.Timeout:
                return {"result": "TimeOut", "details": {"url": self.url}}
            except requests.exceptions.ConnectionError:
                return {"result": "Connection Error", "details": {"url":
self.url}}
        return {"result": "Not Vulnerable", "details": {}}


def main():
    o = OpenRedirectsDetector()
    # print(o.detect_or("https://bugslayers-cs416-open-redirect.herokuapp.com/?
url=https://www.google.com"))
    # print(o.detect_or("https://medium.com/r/?url=https://phising-
malicious.com"))
    # print(o.detect_or("google.co.in"))

if __name__ == "__main__":
    main()
```

2. Phishing Detection

2.1 detector.py

```
import pickle
import os
from .extractor import WebsiteFeatureExtractor

class PhishingWebsiteDetector:
    def __init__(self,
rel_model_path="./phishing_detection_model/phishing_website_detection_model.sav
"):
        self.model =
pickle.load(open(os.path.join(os.path.dirname(os.path.abspath(__file__)),
rel_model_path), "rb"))
        self.wfe = None

        self.interpretation = {
            -1: "Phishing",
            1: "Legitimate",
            0: "Suspicious"
        }

    def detect_phishing(self, url):
        self.wfe = WebsiteFeatureExtractor(url)

        precheck_res = self.__prechecks()

        if precheck_res["passed"]:
            feature_vec = self.wfe.extract_features()
            res = self.interpretation[self.model.predict([feature_vec])[0]]
            response = {
                "result": res
            }


            details = {
                "prechecks": "passed",
                "features": []
            }
            for feature, desc, val in zip(self.wfe.feature_names,
self.wfe.feature_desc, feature_vec):
                details["features"].append({
                    "feature": feature,
                    "desc": desc,
                    "value": self.interpretation[val]
                })
            response["details"] = details

            return response
        else:
            return {"result": "Phishing", "details": {"prechecks": "failed",
"failed_prechecks": precheck_res["details"]["failed_prechecks"]}}

    def __prechecks(self):
        res = {
            "passed": True,
            "details": {
                "failed_prechecks": []
            }
        }
        if self.wfe.soup == -999:
            res["passed"] = False
            res["details"]["failed_prechecks"].append("Unable to fetch
website")
        else:
            if len(self.wfe.soup.find_all(["html"])) == 0:
                res["passed"] = False
                res["details"]["failed_prechecks"].append("No HTML Tag")

        return res
```

2.2 extractor.py

```python
import re
import os
import ssl, socket
import numpy as np
import datetime
import requests
import tldextract
import whois
import ipaddress
from googlesearch import search
from bs4 import BeautifulSoup
from dotenv import load_dotenv

import time

###########################################################################
############################ Feature Extractor ############################
###########################################################################

# Features to be extracted & stored in the following order:

# Index  Feature
# -----  -------------------------
# 0      SSLfinal_State
# 1      URL_of_Anchor
# 2      Links_in_tags
# 3      web_traffic
# 4      Prefix_Suffix
# 5      having_Sub_Domain
# 6      SFH
# 7      Request_URL
# 8      Links_pointing_to_page
# 9      Google_Index
# 10     URL_Length
# 11     DNSRecord
# 12     Domain_registeration_length
# 13     having_IP_Address
# 14     HTTPS_token
# 15     Page_Rank
# 16     age_of_domain
# 17     popUpWidnow
# 18     Iframe
# 19     on_mouseover
# --------------------------------

class WebsiteFeatureExtractor:
    def __init__(self, url):

        load_dotenv()

        # Converts the given URL into standard format
        self.url = url
        if not re.match(r"^https?", url):
            self.url = "https://" + url
        self.soup = self.getParsedResponse()
        self.subdomain, self.domain, self.suffix = self.parseDomain()
        self.whois_response = whois.whois(self.domain + "." + self.suffix)

        self.feature_names = [
            "SSLfinal_State",
            "URL_of_Anchor",
            "Links_in_tags",
            "web_traffic",
            "Prefix_Suffix",
            "having_Sub_Domain",
            "SFH",
            "Request_URL",
            "Links_pointing_to_page",
            "Google_Index",
            "URL_Length",
            "DNSRecord",
            "Domain_registeration_length",
            "having_IP_Address",
            "HTTPS_token",
            "Page_Rank",
            "age_of_domain",
            "popUpWidnow",
            "Iframe",
            "on_mouseover"
        ]
        self.feature_desc = [
            "Checks the age & issuer of SSL Certificate for website",
            "% of URLs in pointing to different domains or not to any webpage",
            "% of links in <script>, <link> & <meta> tags with different domains",
            "Popularity of a website using ranks from the Alexa database",
            "Prefixes or suffixes separated by (-) to the domain name",
            "Existence of multiple subdomains in the URL",
            "Check if the Server Form Handler has about:blank",
            "% of external objects within a webpage loaded from different domain",
            "Number of backlinks pointing to the page",
            "Check if the page is in Google's index or not",
            "Length of the URL (longish URLs are considered phishy)",
            "Existence of a DNS record for the webpage in the WHOIS database",
            "Registration period of domain & time until expiration",
            "Presence of an IP address (decimal/hex) in the domain part of URL",
            "Existence of HTTPS Token in the Domain Part of the URL",
            "Google's PageRank value for a webpage",
            "Time since creation of domain name of the website",
            "Existence of popups such as prompt() or alert() in the webpage",
            "Presence of <iframe> in a webpage to display additional webpages",
            "Use of onmouseover() event to change address bar contents"
        ]
        self.features = np.zeros(20, dtype=int)

    def extract_features(self):
        # Extract all features & return the feature vector
        # s = time.time()
        self.features[0] = self.checkSSLfinalState()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[1] = self.checkUrlOfAnchor()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[2] = self.checkLinksInTags()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[3] = self.checkWebTraffic()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[4] = self.checkPrefixSuffix()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[5] = self.checkHavingSubdomain()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[6] = self.checkSFH()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[7] = self.checkRequestUrl()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[8] = self.checkLinksPointingToPage()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[9] = self.checkGoogleIndex()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[10] = self.checkUrlLength()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[11] = self.checkDNSRecord()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[12] = self.checkDomainRegistrationLength()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[13] = self.checkHavingIPAddress()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[14] = self.checkHTTPSToken()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[15] = self.checkPageRank()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[16] = self.checkAgeOfDomain()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[17] = self.checkPopUpWindow()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[18] = self.checkIframe()
        # print("%.4f s" % (time.time() - s))
        # s=time.time()
        self.features[19] = self.checkOnMouseOver()
        # print("%.4f s" % (time.time() - s))
```

```python
        return self.features


    ###################################################################
    ########################## Utility Functions ######################
    ###################################################################

    def getFeatureNames(self):
        return self.feature_names

    def getParsedResponse(self, url=None):
        url = url or self.url

        # Stores the response of the given URL
        try:
            response = requests.get(url, timeout=5)
            soup = BeautifulSoup(response.text, 'html.parser')
        except Exception:
            response = ""
            soup = -999

        return soup

    def parseDomain(self, url=None):
        url = url or self.url

        # Extracts domain from the given URL
        subdomain, domain, suffix = tldextract.extract(url)
        return subdomain, domain, suffix

    def getWhoisResponse(self, url=None):
        if url:
            _, domain, suffix = tldextract.extract(url)
            domain_name = domain + "." + suffix
            return whois.whois(domain_name)
        else:
            return self.whois_response

    def getDomainAndParsedResponse(self, url=None):
        if url == None:
            domain = self.domain
            soup = self.soup
        else:
            domain = self.parseDomain(url)[1]
            soup = self.getParsedResponse(url)
        return domain, soup

    def parseDatetimeString(self, datetime_str):
        return datetime.datetime.strptime(datetime_str, "%b %d %H:%M:%S %Y %Z")

    ###################################################################
    #################### Feature Extraction Functions #################
    ###################################################################

    # 0. SSLfinal_State
    def checkSSLfinalState(self, url=None):
        hostname = ".".join(self.parseDomain(url)[1:])
        reputed_issuers = ["IdenTrust", "DigiCert Inc", "Sectigo Limited",
"GoDaddy.com, Inc.", "GlobalSign nv-sa", "Actalis S.p.A.", "Entrust, Inc.",
"Google Trust Services", "Microsoft Corporation", "Amazon"]
        try:
            ctx = ssl.create_default_context()
            with ctx.wrap_socket(socket.socket(), server_hostname=hostname) as
s:
                s.settimeout(5)
                s.connect((hostname, 443))
                cert = s.getpeercert()
            issuer = dict(x[0] for x in cert['issuer'])["organizationName"]
            issued_on = self.parseDatetimeString(cert["notBefore"])
            expires_on = self.parseDatetimeString(cert["notAfter"])
            cert_valid = (expires_on - datetime.datetime.today()).days > 0 and
(datetime.datetime.today() - issued_on).days > 0

            if cert_valid:
                if issuer in reputed_issuers:
                    return 1
                else:
                    return 0
            else:
                return -1
        except Exception:
            return -1


    # 1. URL_of_Anchor
    def checkUrlOfAnchor(self, url=None):
        domain, soup = self.getDomainAndParsedResponse(url)
        i = 0
        unsafe=0
        if soup == -999:
            return -1
        else:
            for a in soup.find_all('a', href=True):
                if (not domain in a['href']) or a['href'].startswith("#") or
"javascript" in a['href'].lower():
                    unsafe += 1
                i += 1
            try:
                percentage = unsafe / float(i) * 100
            except Exception:
                # No <a> tags with href found
                return 1

            if percentage < 31.0:
                return 1
            elif percentage <= 67.0:
                return 0
            else:
                return -1

    # 2. Links_in_tags
    def checkLinksInTags(self, url=None):
        domain, soup = self.getDomainAndParsedResponse(url)
        i=0
        success = 0
        if soup == -999:
            return -1
        else:
            for link in soup.find_all('link', href= True):
                if domain not in link['href']:
                    success = success + 1
                i = i+1

            for script in soup.find_all('script', src= True):
                if domain not in script['src']:
                    success = success + 1
                i = i+1
            try:
                percentage = success / float(i) * 100
            except Exception:
                return 1

            if percentage < 17.0 :
                return 1
            elif percentage <= 81.0:
                return 0
            else :
                return -1

    # 3. web_traffic
    def checkWebTraffic(self, url=None):
        url = url or self.url
        try:
            r = requests.get('http://tools.mercenie.com/alexa-rank-
checker/api/?format=json&urls=' + url, timeout=2)
            data = r.json()
            rank = int(data['alexaranks']['first']['alexarank']['0'])

            if rank < 100000:
                return 1
            else:
                return 0
        except Exception:
            return -1

    # 4. Prefix_Suffix
    def checkPrefixSuffix(self, url=None):
        domain = self.getDomainAndParsedResponse(url)[0]
        if "-" in domain:
            return -1
        else:
            return 1

    # 5. having_Sub_Domain
    def checkHavingSubdomain(self, url=None):
        if url == None:
            subdomains = self.subdomain.split(".")
        else:
            subdomains = tldextract(url)[0].split(".")

        if "www" in subdomains:
            subdomains.remove("www")
        if len(subdomains) == 0:
            return 1
        elif len(subdomains) == 1:
            if subdomains[0] == "":
                return 1
            else:
                return 0
        else:
            return -1

    # 6. SFH
    def checkSFH(self, url=None):
        domain, soup = self.getDomainAndParsedResponse(url)
        sfh_empty = 0
        sfh_other_domain = 0
        for form in soup.find_all('form', action= True):
            if form['action'] =="" or form['action'] == "about:blank" :
                sfh_empty += 1
            elif domain not in form['action']:
                sfh_other_domain += 1

        if sfh_empty > 0:
            return -1
        elif sfh_other_domain > 0:
            return 0
        else:
            return 1

    # 7. Request_URL
    def checkRequestUrl(self, url=None):
        domain, soup = self.getDomainAndParsedResponse(url)
        i = 0
        success = 0

        if soup == -999:
            return -1

        else:
            for elem in soup.find_all(["img", "audio", "embed", "iframe",
"video"]):
                # Check for <source> tags within the media elements
                for src in elem.find_all("source"):
                    if domain in src['src']:
                        success += 1
                    i += 1

                # Check for src attribute of the media elements
                if elem.has_attr("src"):
                    if domain in elem['src']:
                        success += 1
                    i += 1

            try:
                percentage = success/float(i) * 100
            except Exception:
                return 1

            if percentage < 22.0 :
                return 1
            elif percentage <= 61.0:
                return 0
            else :
                return -1

    # 8. Links_pointing_to_page
    def checkLinksPointingToPage(self, url=None):
        url = url or self.url
        try:
            data = {"urlo": url}
            r = requests.post("http://tools.mercenie.com/moz-checker/",
data=data, timeout=2)
            soup = BeautifulSoup(r.text, 'html.parser')

            for row in soup.find_all("tr"):
                row_data = row.find_all("td")
                if len(row_data) == 2 and row_data[0].text == "Number of
Backlinks":
                    num_backlinks = int(row_data[1].text)

            if num_backlinks == 0:
                return -1
            elif num_backlinks <= 2:
                return 0
            else:
                return 1

        except Exception:
            return -1

    # 9. Google_Index
    def checkGoogleIndex(self, url=None):
        url = url or self.url
        try:
            results = search(url)
            if url in results:
                return 1
            else:
                return -1
        except Exception:
            return -1

    # 10. URL_Length
    def checkUrlLength(self, url=None):
        url = url or self.url
        if len(url) < 54:
            return 1
        elif len(url) >= 54 and len(url) <= 75:
            return 0
        else:
            return -1

    # 11. DNSRecord
    def checkDNSRecord(self, url=None):
        whois_response = self.getWhoisResponse(url)
        if "domain_name" not in whois_response.keys() or
whois_response["domain_name"] == None:
            return -1
        else:
            return 1

    # 12. Domain_registeration_length
    def checkDomainRegistrationLength(self, url=None):
        if self.checkDNSRecord(url) == -1:
            return -1

        whois_response = self.getWhoisResponse(url)

        # Obtain the domain creation date
        if "creation_date" not in whois_response.keys():
            return -1
        if type(whois_response["creation_date"]) == datetime.datetime:
            creation_date = whois_response["creation_date"]
        elif type(whois_response["creation_date"]) == list:
            creation_date = whois_response["creation_date"][0]

        # Obtain the domain expiration date
        if "expiration_date" not in whois_response.keys():
            return -1
        if type(whois_response["expiration_date"]) == datetime.datetime:
            expiration_date = whois_response["expiration_date"]
        elif type(whois_response["expiration_date"]) == list:
            expiration_date = whois_response["expiration_date"][0]

        reg_len = expiration_date - creation_date
        if reg_len.days <= 365:                     # Less than 1 year
            return -1
        else:
            return 1

    # 13. having_IP_Address
    def checkHavingIPAddress(self, url=None):
        domain= self.getDomainAndParsedResponse(url)[0]
        try:
            resp = ipaddress.ip_address(domain)
            return -1
        except Exception:
            return 1

    # 14. HTTPS_token
    def checkHTTPSToken(self, url=None):
        if url:
            subdomain, domain, _ = tldextract.extract(url)
        else:
            subdomain = self.subdomain
            domain = self.domain

        if "https" in subdomain or "https" in domain:
            return -1
        else:
            return 1

    # 15. Page_Rank
    def checkPageRank(self, url=None):
        if url:
            _, domain, suffix = self.parseDomain(url)
            domain_name = domain + "." + suffix
        else:
            domain_name = self.domain + "." + self.suffix

        try:
            page_rank_url = "https://openpagerank.com/api/v1.0/getPageRank?
domains[]=" + domain_name
            headers = {"API-OPR": os.getenv("OPR_API_KEY")}
            r = requests.get(page_rank_url, headers=headers, timeout=2).json()
            pr = float(r["response"][0]["page_rank_decimal"])/10
        except Exception:
            return -1

        if pr < 0.2:
            return -1
        else:
            return 1

    # 16. age_of_domain
    def checkAgeOfDomain(self, url=None):
        if self.checkDNSRecord(url) == -1:
            return -1

        whois_response = self.getWhoisResponse(url)

        # Obtain the domain creation date
        if "creation_date" not in whois_response.keys():
            return -1
        if type(whois_response["creation_date"]) == datetime.datetime:
            creation_date = whois_response["creation_date"]
        elif type(whois_response["creation_date"]) == list:
            creation_date = whois_response["creation_date"][0]

        # Obtain today's date
        today = datetime.datetime.today()

        age = (today - creation_date).days
        if age >= 182:
            return 1
        else:
            return -1

    # 17. popUpWindow
    def checkPopUpWindow(self, url=None):
        soup = self.getDomainAndParsedResponse(url)[1]
        if len(re.findall(r"(prompt\()|(alert\()", str(soup))) > 0:
            return -1
        else:
            return 1

    # 18. Iframe
    def checkIframe(self, url=None):
        soup = self.getDomainAndParsedResponse(url)[1]
        if len(soup.find_all("iframe", frameborder="0")) > 0:
            return -1
        else:
            return 1

    # 19. on_mouseover
    def checkOnMouseOver(self, url=None):
        soup = self.getDomainAndParsedResponse(url)[1]
        if len(re.findall(r"onmouseover", str(soup))) > 0:
            return -1
        else:
            return 1
```

3. TLS Certificate detection

3.1 cert_checker.py

```python
import datetime
import select
import socket
import sys
import OpenSSL
import json

class Domain(str):
    def __new__(cls, domain):
        host = domain
        port = 443
        connection_host = host
        result = str.__new__(cls, host)
        result.host = host
        result.connection_host = connection_host
        result.port = port
        return result

class CertChecker:
    def __init__(self):
        pass

    def get_cert_from_domain(self, domain):
        ctx = OpenSSL.SSL.Context(OpenSSL.SSL.SSLv23_METHOD)
        sock = socket.socket()
        sock.settimeout(5)
        wrapped_sock = OpenSSL.SSL.Connection(ctx, sock)
        wrapped_sock.set_tlsext_host_name(domain.host.encode('ascii'))
        wrapped_sock.connect((domain.connection_host, 443))
        while True:
            try:
                wrapped_sock.do_handshake()
                break
            except OpenSSL.SSL.WantReadError:
                select.select([wrapped_sock], [], [])
        return wrapped_sock.get_peer_cert_chain()

    def get_domain_certs(self, domains):
        domain = Domain(domains)
        try:
            data = self.get_cert_from_domain(domain)
        except Exception as e:
            data = e
        return data

    def validate_cert(self, cert_chain):
        msgs = []
        ctx = OpenSSL.SSL.Context(OpenSSL.SSL.SSLv23_METHOD)
        ctx.set_default_verify_paths()
        cert_store = ctx.get_cert_store()
        for index, cert in reversed(list(enumerate(cert_chain))):
            sc = OpenSSL.crypto.X509StoreContext(cert_store, cert)
            try:
                sc.verify_certificate()
            except OpenSSL.crypto.X509StoreContextError as e:
                msgs.append(
                    ('error', "Validation error '%s'." % e))
            if index > 0:
                cert_store.add_cert(cert)
        return msgs


    def check(self, domain,domain_certs,utcnow):
        msgs = []
        results = []

        earliest_expiration = None
        if domain_certs is None: return (msgs, earliest_expiration, None)
        if isinstance(domain_certs, Exception):
            domain_certs =
"".join(traceback.format_exception_only(type(domain_certs),domain_certs)).strip
()
        msgs = self.validate_cert(domain_certs)
        for i, cert in enumerate(domain_certs):
            result = {}
            subject = cert.get_subject().commonName
            result["subject"] = subject
            expires =
datetime.datetime.strptime(cert.get_notAfter().decode('ascii'),
'%Y%m%d%H%M%SZ')
            result["expires"] = str(expires)

            if expires:
                if earliest_expiration is None or expires <
earliest_expiration:
                    earliest_expiration = expires
            issued_level = "info"
            issuer = cert.get_issuer().commonName
            if issuer:
                if issuer.lower() == "happy hacker fake ca":
                    issued_level = "error"
            else:
                issued_level = 'warning'
            msgs.append((issued_level, "Issued by: %s (subject: %s)" % (issuer,
subject)))
            result["issuer"] = issuer

            results.append(result)

            if i < len(domain_certs) - 1:
                sign_cert = domain_certs[i+1]
                subject = sign_cert.get_subject().commonName
                if issuer != subject:
                    msgs.append(
                        ('error', "The certificate sign chain subject '%s'
doesn't match the issuer '%s'." % (subject, issuer)))

            sig_alg = cert.get_signature_algorithm()
            if sig_alg.startswith(b'sha1'):
                msgs.append(('error', "Unsecure signature algorithm %s
(subject: %s)" % (sig_alg, subject)))
            if expires < utcnow:
                msgs.append(
                    ('error', "The certificate has expired on %s (subject: %s)"
% (expires, subject)))
            elif expires < (utcnow + datetime.timedelta(days=15)):
                msgs.append(
                    ('warning', "The certificate expires on %s (%s) (subject:
%s)" % (
                        expires, expires - utcnow, subject)))
            else:
                delta = ((expires - utcnow) // 60 // 10 ** 6) * 60 * 10 ** 6
                msgs.append(
                    ('info', "Valid until %s (%s)." % (expires, delta)))

        return (msgs, earliest_expiration, results)


    def checkCertChain(self, domain):
        domain = domain.replace("http://","")
        domain = domain.replace("https://","")
        domain = domain.split('/')[0]
        domain_certs = self.get_domain_certs(domain)

        if isinstance(domain_certs, Exception):
            output = {
                "result": "Invalid",
                "errors": ["Unable to obtain certficate chain"],
                "warnings": [],
                "details": []
            }
            return output

        utcnow = datetime.datetime.now()
        (msgs, earliest_expiration, results) =
self.check(domain,domain_certs,utcnow)

        warnings = []
        output = {}
        output["details"] = results
        errors = []
        for level, msg in msgs:
            if level == 'error':
                errors.append(msg)
            elif level == 'warning':
                warnings.append(msg)

        output["errors"] = errors
        output["warnings"] = warnings

        if len(errors) > 0:
            output["result"] = "Invalid"
        elif len(warnings) > 0:
            output["result"] = "Valid (with Warnings)"
        else: output["result"] = "Valid"
        return output
```

4. XSS Detection

4.1 main.py

```python
import uvicorn
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi
from starlette.middleware.cors import CORSMiddleware
from fastapi.responses import RedirectResponse
import v1

app = FastAPI()

app.add_middleware(
    CORSMiddleware, allow_origins=["http://localhost:8000"], allow_methods=["*"], allow_headers=["*"]
)

@app.get("/", tags=["root"])
def get_root():
    return RedirectResponse("/v1/")


app.include_router(
    v1.router,
    prefix="/v1",
    tags=["vulnerability scanning (version 1)"],
    responses={404: {"description": "Not found"}},
)


def custom_openapi():
    if app.openapi_schema:
        return app.openapi_schema
    openapi_schema = get_openapi(
        title="WAVS: Web App Vulnerability Scanner",
        version="0.0.1",
        description="""WAVS (Web App Vulnerability Scanner) is a tool to scan & test URLs for certain vulnerabilities &
        security issues by simply inspecting the corresponding client-side website. The overall system would include 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.
        \nWe identify & assess the following security issues that a website may suffer from: _Absence of Valid TLS Certificates_,
        _Cross-Site Scripting (XSS)_, _Potential Phishing Attempts_ & _Open Redirection_
        """,
        routes=app.routes,
    )
    openapi_schema["info"]["x-logo"] = {
        "url": "https://fastapi.tiangolo.com/img/logo-margin/logo-teal.png"
    }
    app.openapi_schema = openapi_schema
    return app.openapi_schema


app.openapi = custom_openapi

if __name__ == "__main__":
    uvicorn.run("main:app", host="localhost", port=9000)
```

Proxy-code

1. wavs_proxy.js

```
const hoxy = require('hoxy');
const ejs = require('ejs');
const fs = require('fs');
const { program } = require('commander');

program
  .option('-s, --scan <type>', 'Scan Type ("full" or "selective")', 'full')
  .option('-t, --tls_cert', 'Enable scan for SSL/TLS Certificates')
  .option('-x, --xss', 'Enable scan for Cross-Site Scripting')
  .option('-p, --phishing', 'Enable scan for potential Phishing')
  .option('-o, --open_redirect', 'Enable scan for Open Redirects')

program.parse(process.argv);
const options = program.opts();

scan_options = {
  tls_cert: false,
  xss: false,
  phishing: false,
  open_redirect: false
};

if (options.scan === 'full') {
  scan_options = {
    tls_cert: true,
    xss: true,
    phishing: true,
    open_redirect: true
  }
}
else if (options.scan === 'selective') {
  if (options.tls_cert) {
    scan_options["tls_cert"] = true;
  }
  if (options.xss) {
    scan_options["xss"] = true;
  }
  if (options.phishing) {
    scan_options["phishing"] = true;
  }
  if (options.open_redirect) {
    scan_options["open_redirect"] = true;
  }
}


homepage_str = "";
```

```
fs.readFile("./templates/homepage.html", (error, data) => {
  if(error) {
      throw error;
  }
  homepage_str = data.toString();
});

var server_port = 9000;
var proxy_port = 8000;


// Proxy server to direct requests to Virtual Server
var proxy = hoxy.createServer().listen(proxy_port, function() {
  console.log('The Proxy is listening on port', proxy_port, "\n");
});

// Intercept & modify request

// If root path, display landing page template
proxy.intercept({
  phase: 'request',
  url: '/'
}, (req, res) => {
  res.headers = {'Content-Type': 'text/html'};
  res.statusCode = 200;
  // res.string = "<html><body><h2>Landing Page!</h2></body></html>";
  res.string = homepage_str;
})

// For all other requests, redirect to virtual server
proxy.intercept({
  phase: 'request'
}, (req, res) => {
  query_url = req.url.slice(1);

  // Modify the request to call the required endpoint with correct query param
  req.method = 'POST';
  req.hostname = 'localhost';          // Domain of Virtual Server
  req.port = server_port;              // Port on which Virtual Server is
running
  req.url = '/v1/scan/?url=' + query_url  // Appropriate endpoint on Virtual
Server
  req.json = scan_options
})

////////////////////////////////////////////////////////////////////////////////
///////////

// Intercept & modify response
proxy.intercept({
  phase: 'response',
  as: 'json'
}, async (req, res) => {
  // console.log("Response intercepted by proxy:", res.json)

  // Use JSON to manipulate template...
  html_str = await ejs.renderFile("templates/dashboard.ejs", res.json);
  // Now return the filled template
  res.headers = {'Content-Type': 'text/html'};
  res.statusCode = 200;
  res.string = html_str;
  // console.log("Modified Response:", res.string,
"\n======================================================")
})
```

2. templates

2.1 dashboard.ejs

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://fonts.googleapis.com/css2?
family=RocknRoll+One&display=swap" rel="stylesheet">
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?
family=Material+Icons">
<style>
* {
  box-sizing: border-box;
  font-family: 'RocknRoll One', sans-serif;
}

.column-img {
  float: left;
  width: 25%;
  border: 30px;
  height: 300px;
  position: relative;
  /* left: 10px; */
  align-items: center;
  text-align: center;
  display: block;
}
.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
  margin-top: 30px;
  width: 80%;
  height: 80%;
}
.column {
  float: left;
  width: 75%;
  padding: 20px;
  border: 30px;
  height: 300px;
  position: relative;
}

.row{
    margin: 10px;
}
/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}
.bg-yellow {
    background: #E3AE57;
}

.bg-skin {
    background: #f1e2d7;
}
.bg-red {
    background: #E9724C;
}
.text-white {
    color: #FFF;
}
.text-dark {
    color: #000000;
}
h2 {
    text-align: center;
}
.heading {
    text-align: center;
}
.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  padding-top: 100px; /* Location of the box */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}

/* Modal Content */
.modal-content {
  position: relative;
  background-color: #fefefe;
  margin: auto;
  padding: 0;
  border: 1px solid #888;
  width: 80%;
  box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2),0 6px 20px 0 rgba(0,0,0,0.19);
  -webkit-animation-name: animatetop;
  -webkit-animation-duration: 0.4s;
  animation-name: animatetop;
  animation-duration: 0.4s
}

/* Add Animation */
@-webkit-keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}
```

```
@keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}

/* The Close Button */
.close {
  color: white;
  float: right;
  font-size: 28px;
  font-weight: bold;
}

.close:hover,
.close:focus {
  color: #000;
  text-decoration: none;
  cursor: pointer;
}

.modal-header {
  padding: 2px 16px;
  background-color: #E3AE57;
  color: white;
}

.modal-body {padding: 2px 16px;}

.modal-footer {
  padding: 2px 16px;
  background-color:  #E9724C;
  color: white;
}

button {
    display: inline-block;
    /* width: ; */
    transition: all 0.5s;
    cursor: pointer;
    /* margin: 5px; */
}
#proceed {
    background-color:#000;
    border-radius: 10px;
    border: 3px double #000;
    color: #f1e2d7;
    text-align: center;
    /* font-size: inherit; */
    /* padding: 10px; */
    padding-left: 20px;
    padding-right: 20px;

}
#phish_param {
    background-color:#b68434;
    /* border-radius: 10px; */
    border: 2px double #cccccc;
    border-radius: 5px;
    padding: 5pt;
    /* color: #eeeeee; */
    text-align: center;
    font-size: inherit;
}

#xss_details {
    background-color:#c9b4a5;

    border: 2px double;
    border-color: #121212;;
    border-radius: 5px;
    padding: 5pt;
    color: #121212;
    text-align: center;
    font-size: inherit;
}

#tls_cert_details {
    background-color:#b85230;
    /* border-radius: 10px; */
    border: 2px double #cccccc;
    border-radius: 5px;
    padding: 5pt;
    /* color: #eeeeee; */
    text-align: center;
    font-size: inherit;
}
button span {
    cursor: pointer;
    display: inline-block;
    position: relative;
    transition: 0.5s;
}
button span:after {
    content: '\00bb';
    position: absolute;
    opacity: 0;
    right: -20px;
    transition: 0.5s;
}
/* button:hover {
background-color: #f7c2f9;
} */
button:hover span {
    padding-right: 25px;
}
#proceed:hover span {
    padding-left: 25px;
}
button:hover span:after {
    opacity: 1;
    right: 0;
}

table {
  border-collapse: collapse;
  width: 100%;
  margin-bottom: 10px;
}

td, th {
  border: 3px solid #f1e2d7;
  text-align: center;
  padding: 8px;
}

#footer {
    position: absolute;
    /* display: inline; */
    padding: 10px 10px 10px 10px;
    bottom: 20px;
    width: 95%;
    /* Height of the footer*/
    height: 40px;
    /* background: grey; */
}
p {
    padding: 10px 10px 0px 10px;
}

i {
    vertical-align: middle;
}

a {
  color: #eee;
  text-decoration: none;
}
</style>

<title>WAVD Scan Results</title>
</head>
<body>

<div class="heading">
    <h1>WAVD Dashboard: Scan Results</h1>
    <h3>URL: <%= url %></h3>
</div>



<% if (locals.phishing) { %>
<div class="row">
    <div class="column-img">
      <img src="https://i.ibb.co/rtpgqVD/phishing.jpg" class="center">
    </div>
    <div class="column bg-yellow text-white" style="text-align: center;">
        <h2>Potential Phishing Attempts</h2>
        <% if (phishing.result === "Legitimate") { %>
            <p><i class="material-icons" style="color: green; ">done</i> <b>Legitimate:</b> This website does not seem to have any major characteristics of a phishing domain</p>
        <% } %>
        <% if (phishing.result === "Suspicious") { %>
            <p><i class="material-icons" style="color: yellow;">error</i>
<b>Suspicious:</b> This website may be phishy, although we do not have
conclusive evidence</p>
        <% } %>
        <% if (phishing.result === "Phishing") { %>
            <p><i class="material-icons" style="color: red;">dangerous</i>
<b>Phishing:</b> This website seems to be <i>phishy</i>! We recommend not to
visit it</p>
        <% } %>


      <!-- Trigger/Open The Modal -->
        <button id="phish_param"><span class="text-white">Click here for
Detailed Analysis</span></button>
        <div id="footer" style="text-align: center;">
            <b>Quick Links:</b> <a target="_blank"
href="https://www.imperva.com/learn/application-security/phishing-attack-
scam">Introduction to Phishing Attacks</a> | <a target="_blank"
href="https://www.globalsign.com/en/blog/how-to-spot-a-fake-website">How to
Spot a Phishing Website</a>
        </div>
    </div>
</div>
<% } %>

<% if (locals.tls_cert) { %>
<div class="row">
  <div class="column bg-red text-white" style="text-align: center;">
    <h2>SSL/TLS Certificate Validation</h2>
    <% if (tls_cert.result === "Valid") { %>
        <p><i class="material-icons" style="color: green; ">done</i> <b>Valid:
</b> This website possesses a valid SSL/TLS certificate</p>
    <% } %>
    <% if (tls_cert.result === "Valid (with Warnings)") { %>
        <p><i class="material-icons" style="color: yellow; ">error</i> <b>Valid
(with Warnings):</b> This website has valid SSL/TLS certificate, but the
issuing authority or expiry may be questionable</p>
    <% } %>
    <% if (tls_cert.result === "Invalid") { %>
        <p><i class="material-icons" style="color: rgb(214, 0,
0);">dangerous</i> <b>Invalid:</b> This website does not have valid SSL/TLS
certificate</p>
    <% } %>

    <button id="tls_cert_details"><span class="text-white">Click here for
Certificate Details</span></button>
    <div id="footer">
        <b>Quick Links:</b> <a target="_blank"
href="https://protonmail.com/blog/tls-ssl-certificate/">Working of SSL/TLS
Certificates</a> | <a target="_blank" href="https://www.venafi.com/blog/how-do-
certificate-chains-work">Certificate Chains</a>
    </div>
  </div>
  <div class="column-img">
    <img src="https://i.ibb.co/W3rfyHc/tls.png"  class="center">
  </div>
</div>
<% } %>
<% if (locals.open_redir) { %>
<div class="row">
  <div class="column-img">
    <img src="https://i.ibb.co/RDWx8Gs/or.jpg" class="center">
  </div>
  <div class="column text-white" style="background-color:#000; text-align:
center;">
    <h2>Open Redirects</h2>

    <% if (open_redir.result === "Not Vulnerable") { %>
        <p><i class="material-icons" style="color: rgb(75, 255, 75); ">done</i>
<b>Not Vulnerable:</b> This website has not been identified with any Open
Redirect vulnerabilities</p>
    <% } else if (open_redir.result === "Header Based Redirection" ||
open_redir.result === "Open Redirection") { %>
      <p><i class="material-icons" style="color: red; ">dangerous</i>
<b>Vulnerability Detected:</b> This website has been identified with <i><%=
open_redir.result %></i> </p>

      <% if (open_redir.result === "Header Based Redirection") { %>
        <p><b>Query Parameter for Redirection:</b> <%=
open_redir.details.parameter %></p>
      <% } %>
    <% } else { %>
      <p><i class="material-icons" style="color: yellow; ">error</i> <b>Client
Error:</b> <i><%= open_redir.result %></i> </p>
    <% } %>



    <div id="footer">
      <b>Quick Links:</b> <a target="_blank"
href="https://portswigger.net/kb/issues/00500100_open-redirection-
reflected">Open Redirect (Reflected)</a> | <a target="_blank"
href="https://blog.detectify.com/2019/05/16/the-real-impact-of-an-open-
redirect/">Impact of Open Redirects</a>
    </div>
  </div>
</div>
<% } %>

<% if (locals.xss) { %>
<div class="row">
  <div class="column bg-skin" style="text-align: center;">
    <h2>Cross-Site Scripting (XSS)</h2>
    <% if (xss.result === "XSS Not Detected") { %>
        <p><i class="material-icons" style="color: green; ">done</i> <b><%=
xss.result %>:</b> This website has no identifiable client-side XSS
vulnerabilities</p>
    <% } %>
    <% if (xss.result === "XSS Detected") { %>
        <p><i class="material-icons" style="color: red;">dangerous</i> <b><%=
xss.result %>:</b> <i>DOM-based XSS</i> vulnerability has been detected in this
website</p>
        <button id="xss_details"><span>Click here for Details</span></button>
    <% } %>
    <div id="footer">
      <b>Quick Links:</b> <a style="color: #121212;" target="_blank"
```

```
href="https://portswigger.net/web-security/cross-site-scripting">Types of
XSS</a> | <a style="color: #121212;" target="_blank"
href="https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Preve
ntion_Cheat_Sheet.html">XSS Prevention</a>
    </div>
  </div>
  <div class="column-img">
    <img src="https://i.ibb.co/C83FMKp/xss.jpg" class="center">
  </div>
</div>
<% } %>

<% if (locals.phishing) { %>
<div id="phish_modal" class="modal">

    <!-- Modal content -->
    <div class="modal-content text-dark">
      <div class="modal-header">
        <span class="close" id="phish_modal_close">&times;</span>
        <h2>Phishing Detection Features</h2>
      </div>
      <div class="modal-body" style="text-align: center;">
        <!-- <p>Some text in the Modal Body</p> -->
        <% if (phishing.details.prechecks === "passed") { %>
            <table>
                <tr>
                <th>Paramter</th>
                <th>Description</th>
                <th>Status</th>
                </tr>
                <% for (let i = 0; i < phishing.details.features.length; i++) {
%>
                    <tr>
                        <td><%= phishing.details.features[i].feature %></td>
                        <td><%= phishing.details.features[i].desc %></td>
                        <td><%= phishing.details.features[i].value %></td>
                    </tr>
                <% } %>
            </table>
        <% } else { %>
            <h3>Failed Prechecks:</h3>

            <% for (let i = 0; i < phishing.details.failed_prechecks.length;
i++) { %>
            <p><%= phishing.details.failed_prechecks[i] %></p>
            <% } %>

        <% } %>
      </div>
      <div class="modal-footer">
        <!-- <h3>M</h3> -->
      </div>
    </div>
</div>
<% } %>

<% if (locals.tls_cert) { %>
<div id="tls_modal" class="modal">
```

```
    <!-- Modal content -->
    <div class="modal-content text-dark">
      <div class="modal-header">
        <span class="close" id="tls_modal_close">&times;</span>
        <h2>SSL/TLS Certificate Details</h2>
      </div>
      <div class="modal-body" style="text-align: center; padding-top: 10px">
        <h3>Certificate Chain</h3>
        <% for (let i = 0; i < tls_cert.details.length; i++) { %>
            <table>
                <tr>
                    <td><b>Subject</b></td>
                    <td><%= tls_cert.details[i].subject %></td>
                </tr>
                <tr>
                    <td><b>Expires On</b></td>
                    <td><%= tls_cert.details[i].expires %></td>
                </tr>
                <tr>
                    <td><b>Issuer</b></td>
                    <td><%= tls_cert.details[i].issuer %></td>
                </tr>
            </table>
            <br>
        <% } %>

        <div style="text-align: center;">
            <% if (tls_cert.warnings.length > 0) { %>
                <hr>
                <h3><i class="material-icons" style="color: rgb(255, 186, 58);
">error</i>Warnings</h3>

                <% for (let i = 0; i < tls_cert.warnings.length; i++) { %>
                    <p><%= tls_cert.warnings[i] %></p>
                <% } %>
            <% } %>

            <% if (tls_cert.errors.length > 0) { %>
                <hr>
                <h3><i class="material-icons" style="color: red;
">dangerous</i>Errors</h3>

                <% for (let i = 0; i < tls_cert.errors.length; i++) { %>
                    <p><%= tls_cert.errors[i] %></p>
                <% } %>
            <% } %>
        </div>
      </div>
      <div class="modal-footer">
    </div>
  </div>
</div>
<% } %>


<% if (locals.xss) { %>
  <div id="xss_modal" class="modal">
```

```
      <div class="modal-content text-dark">
        <div class="modal-header">
          <span class="close" id="xss_modal_close">&times;</span>
          <h2>XSS Details</h2>
        </div>
        <div class="modal-body" style="text-align: center; padding-top: 10px">

          <h3>Details of the Forms detected with XSS Vulnerability:</h3>
          <% for (let i = 0; i < xss.details.length; i++) { %>
            <table>
              <tr>
                <th>Element Name</th>
                <th>Element Type</th>
                <th>Value Injected</th>
              </tr>
              <% for (let j = 0; j < xss.details[i].inputs.length; j++) { %>
                <% if (xss.details[i].inputs[j].name) { %>
                  <tr>
                    <td><%= xss.details[i].inputs[j].name %></td>
                    <td><%= xss.details[i].inputs[j].type %></td>
                    <td><%= xss.details[i].inputs[j].value %></td>
                  </tr>
                <% } %>
              <% } %>
            </table>
            <br/>
            <p><b>Form Action:</b> <%= xss.details[i].action %></p>
            <p><b>Form Submission Method:</b> <%= xss.details[i].method %></p>
            <br/>
            <hr/>
          <% } %>
        </div>
        <div class="modal-footer">
      </div>
    </div>
  </div>
  <% } %>

<div class="heading" style="margin-bottom: 20px;">
    <br>
    <button id="proceed"><span><a target="_blank" style="color: #fff; text-decoration: none;" href="<%= url %>"><h2>Proceed to URL</h2></a></span>
</button>
</div>

<!-- <footer style="width:100%; background-color: #030100; text-align: center; color: #fff; padding: 15px; font-size: 11pt; position: fixed; left: 0; bottom: 0;">
  Created by Rohan Sharma, Shubham Yadav, Dhiraj Mishra
</footer> -->

<% if (locals.phishing) { %>
<script>
    /////////////////////// Phishing Modal //////////////////////////
    // Get the modal
    var phish_modal = document.getElementById("phish_modal");
    // Get the button that opens the modal
    var phish_btn = document.getElementById("phish_param");

    // Get the <span> element that closes the modal
    var phish_modal_close = document.getElementById("phish_modal_close");

    // When the user clicks the button, open the modal
    phish_btn.onclick = function() {
        phish_modal.style.display = "block";
    }

    // When the user clicks on <span> (x), close the modal
    phish_modal_close.onclick = function() {
        phish_modal.style.display = "none";
    }

    // When the user clicks anywhere outside of the modal, close it
    window.onclick = function(event) {
      if (event.target == phish_modal) {
        phish_modal.style.display = "none";
      }
    }
</script>
<% } %>

<% if (locals.tls_cert) { %>
<script>
    //////////////////////// TLS Modal /////////////////////////////
    var tls_modal = document.getElementById("tls_modal");

    // Get the button that opens the modal
    var tls_btn = document.getElementById("tls_cert_details");

    // Get the <span> element that closes the modal
    var tls_modal_close = document.getElementById("tls_modal_close");

    // When the user clicks the button, open the modal
    tls_btn.onclick = function() {
        tls_modal.style.display = "block";
    }

    // When the user clicks on <span> (x), close the modal
    tls_modal_close.onclick = function() {
        tls_modal.style.display = "none";
    }

    // When the user clicks anywhere outside of the modal, close it
    window.onclick = function(event) {
      if (event.target == tls_modal) {
        tls_modal.style.display = "none";
      }
    }
</script>
<% } %>

<% if (locals.xss) { %>
  <script>
      //////////////////////// XSS Modal //////////////////////////////
```

```
      var xss_modal = document.getElementById("xss_modal");

      // Get the button that opens the modal
      var xss_btn = document.getElementById("xss_details");

      // Get the <span> element that closes the modal
      var xss_modal_close = document.getElementById("xss_modal_close");

      // When the user clicks the button, open the modal
      xss_btn.onclick = function() {
          xss_modal.style.display = "block";
      }

      // When the user clicks on <span> (x), close the modal
      xss_modal_close.onclick = function() {
          xss_modal.style.display = "none";
      }

      // When the user clicks anywhere outside of the modal, close it
      window.onclick = function(event) {
        if (event.target == xss_modal) {
          xss_modal.style.display = "none";
        }
      }
  </script>
  <% } %>

</body>
</html>
```

2.2 homepage.html

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://fonts.googleapis.com/css2?
family=RocknRoll+One&display=swap" rel="stylesheet">
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?
family=Material+Icons">
<style>
* {
  box-sizing: border-box;
  font-family: 'RocknRoll One', sans-serif;
}

.heading {
    text-align: center;
    font-size: 40px;
    margin-bottom: 0;
    /* margin: 0; */
    text-shadow: 4px 4px #645f5f;
}

.subheading {
    text-align: center;
    margin-top: -20pt;
    -webkit-text-stroke: 0.05px #454545;
    /* text-shadow: 2px 1px #000; */
    /* text-shadow: -1px 1px 0 #000,
                          1px 1px 0 #000,
                         1px -1px 0 #000,
                        -1px -1px 0 #000; */
}
.text-yellow {
    color: #E3AE57;
}

.text-light {
    color: #fdd0ad;
}
.text-red {
    color: #E9724C;
}
.text-dark {
    color: #000000;
}

.column {
  /* float: center; */
  /* width: 90%; */
  margin: 0 50px 20px 50px;
  padding: 20px;
  /* position: relative; */
  background-color: #fdd0ad;
  border-radius: 5px;
}
#input-container {

  display: flex;
  flex-direction: row; /* Changed to row to align input and button horizontally
*/
  align-items: center;
  justify-content: center;
  margin-top: 50px;
  margin-bottom: 50px;
}

#key {
  width: 400px; /* Increased width of input box */
  padding: 8px 12px;
  border: 2px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
  margin-right: 10px; /* Added margin to create space between input and button
*/
  font-size: 16px;
  text-align: center;
  box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.2); /* Added box shadow */
}

input[type="submit"] {
  background-color: black;
  color: white;
  padding: 10px 16px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
  box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.2); /* Added box shadow */
}


input[type="submit"]:hover {
  background-color: #333; /* Darker shade of black on hover */
}



</style>
<title>WAVD</title>
</head>
<body>

<div class="heading" style="margin-top:-40px">
    <h1><span class="text-dark">W</span>
        <span class="text-dark">A</span>
        <span class="text-dark">V</span>
        <span class="text-dark">D</span></h1>
</div>
<div class="subheading">
    <h2><span class="text-dark">Potential Phishing Attempt</span>&emsp; &ensp;
        <span class="text-dark">SSL/TLS Certificate Validation</span>&emsp;
&ensp;
        <span class="text-dark">Open Redirects</span>&emsp; &ensp;
        <span class="text-dark">Cross-Site Scripting</span>
    </h2>
</div>
<div id="input-container">
    <input type="text" name="key" id="key">
    <input type="submit" value="Go" onclick="changeLocation(); return false;">
</div>
<div style="align-items: center; width: 100%; justify-content: center;">
    <div class="column">
        <h3 style="margin-top: 0">About:</h3>
        <p style="font-size: 11pt">
            <b>WAVD (Web App Vulnerability Detector)</b> is a tool to scan &
test URLs for certain vulnerabilities &
            security issues by simply inspecting the corresponding client-side
website. The overall system includes a
            virtual server with modules for detecting the different
vulnerabilities, along with a proxy server, to direct
            requests from a user's browser to the WAVD virtual server to scan
the URL for potential threats & return a
            dashboard containing the scan results.
            <br><br>
            WAVD scans & detects the following security issues in a website:
Absence of Valid SSL/TLS Certificates,
            Cross-Site Scripting (XSS), Potential Phishing Attempts & Open
Redirection</p>
    </div>
    <!-- <div class="column" >
        <h3 style="margin-top: 0">Instructions for Use:</h3>
        <p style="font-size: 11pt">
            To perform a scan for any URL, just append '<b>/&lt;URL&gt;</b>' to
the proxy address in the address bar.
            <br>
            <i>(For example, if you wish to perform a scan for
'https://google.co.in', you can append '/https://google.co.in'
            to the address bar)</i>

            <br/><br/>

            WAVD will perform the scan within 10-20 s & return a dashboard
containing the details of the potential vulnerabilities in the website.
        </p>
    </div> -->
</div>

<footer style="width:100%; background-color: #030100; text-align: center;
color: #fff; padding: 15px; font-size: 11pt; position: fixed; left: 0; bottom:
0;">
   Created by Rohan Sharma, Shubham Yadav, Dhiraj Mishra
</footer>

<script>
    function changeLocation() {
    var textboxpw = document.getElementById("key");
    window.open('http://localhost:8000/' + textboxpw.value, '_blank');
}
</script>
</body>
</html>
```

2.3 page.html

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://fonts.googleapis.com/css2?
family=RocknRoll+One&display=swap" rel="stylesheet">
<style>
* {
  box-sizing: border-box;
  font-family: 'RocknRoll One', sans-serif;
}

.column-img {
  float: left;
  width: 25%;
  border: 30px;
  height: 300px;
  position: relative;
  /* left: 10px; */
  align-items: center;
  text-align: center;
  display: block;
}
.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
  margin-top: 30px;
  width: 80%;
  height: 80%;
}
.column {
  float: left;
  width: 75%;
  padding: 20px;
  border: 30px;
  height: 300px;
  position: relative;
}

.row{
    margin: 10px;
}
/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}
.bg-yellow {
    background: #E3AE57;
}

.bg-skin {
    background: #fdd5b7;
}
.bg-red {
    background: #E9724C;
}
.text-white {
    color: #FFF;
}
.text-dark {
    color: #000000;
}
h2 {
    text-align: center;
}
.heading {
    text-align: center;
}
.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  padding-top: 100px; /* Location of the box */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}

/* Modal Content */
.modal-content {
  position: relative;
  background-color: #fefefe;
  margin: auto;
  padding: 0;
  border: 1px solid #888;
  width: 80%;
  box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2),0 6px 20px 0 rgba(0,0,0,0.19);
  -webkit-animation-name: animatetop;
  -webkit-animation-duration: 0.4s;
  animation-name: animatetop;
  animation-duration: 0.4s
}

/* Add Animation */
@-webkit-keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}

@keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}

/* The Close Button */
```

```
.close {
  color: white;
  float: right;
  font-size: 28px;
  font-weight: bold;
}

.close:hover,
.close:focus {
  color: #000;
  text-decoration: none;
  cursor: pointer;
}

.modal-header {
  padding: 2px 16px;
  background-color: #E3AE57;
  color: white;
}

.modal-body {padding: 2px 16px;}

.modal-footer {
  padding: 2px 16px;
  background-color:  #E9724C;
  color: white;
}

button {
    display: inline-block;
    /* width: ; */
    transition: all 0.5s;
    cursor: pointer;
    /* margin: 5px; */
}
#proceed {
    background-color:#000;
    border-radius: 10px;
    border: 3px double #000;
    color: #f1e2d7;
    text-align: center;
    /* font-size: inherit; */
    /* padding: 10px; */
    padding-left: 20px;
    padding-right: 20px;

}
#phish_param {
    background-color:#E3AE57;
    /* border-radius: 10px; */
    border: 0px double #cccccc;
    /* color: #eeeeee; */
    text-align: left;
    font-size: inherit;
}
button span {
    cursor: pointer;
    display: inline-block;
    position: relative;
    transition: 0.5s;
}
button span:after {
    content: '\00bb';
    position: absolute;
    opacity: 0;
    right: -20px;
    transition: 0.5s;
}
/* button:hover {
background-color: #f7c2f9;
} */
button:hover span {
    padding-right: 25px;
}
#proceed:hover span {
    padding-left: 25px;
}
button:hover span:after {
    opacity: 1;
    right: 0;
}

table {
  border-collapse: collapse;
  width: 100%;
  margin-bottom: 10px;
}

td, th {
  border: 3px solid #f1e2d7;
  text-align: center;
  padding: 8px;
}

#footer {
    position: absolute;
    /* display: inline; */
    padding: 10px 10px 0px 10px;
    bottom: 20px;
    width: 70%;
    /* Height of the footer*/
    height: 40px;
    /* background: grey; */
}
p {
    padding: 10px 10px 0px 10px;
}
</style>
<title>Scan Results</title>
</head>
<body>

<div class="heading">
    <h1>Scan Results</h1>
</div>
```

```
<div class="row">
  <div class="column-img">
    <img src="../templates/img/phishing.jpeg" class="center">
  </div>
  <div class="column bg-yellow text-white" style="">
    <h2>Potential Phishing Attempts</h2>
    <p>Some text..</p>

    <!-- Trigger/Open The Modal -->
    <button id="phish_param"><span class="text-white">Click here for detailed analysis</span></button>
    <div id="footer">
      Quick Links
      <ul>
          <li><a href="https://www.imperva.com/learn/application-security/phishing-attack-scam">A Gentle Introduction to Phishing Attacks</a></li>
          <li><a href="https://www.globalsign.com/en/blog/how-to-spot-a-fake-website">How to Spot a Phishing Website</a></li>
      </ul>
    </div>
  </div>
</div>


<div class="row">
  <div class="column bg-red text-white" style="">
    <h2>SSL/TLS Certificate Validation</h2>
    <p>Some text..</p>
    <div id="footer">Quick Links</div>
  </div>
  <div class="column-img">
    <img src="../templates/img/tls.png"  class="center">
  </div>
</div>


<div class="row">
  <div class="column-img">
    <img src="../templates/img/or.jpeg" class="center">
  </div>
  <div class="column text-white" style="background-color:#000;">
    <h2>Open Redirects</h2>
    <p>Some text..</p>
    <div id="footer">Quick Links</div>
  </div>
</div>


<div class="row">
  <div class="column bg-skin" style="">
    <h2>Cross-Site Scripting (XSS)</h2>
    <p>Some text..</p>
    <div id="footer">Quick Links</div>
  </div>
  <div class="column-img">
    <img src="../templates/img/xss.jpeg" class="center">
  </div>
</div>

<div id="myModal" class="modal">

    <!-- Modal content -->
    <div class="modal-content text-dark">
      <div class="modal-header">
        <span class="close">&times;</span>
        <h2>Phishing Detection Parameters</h2>
      </div>
      <div class="modal-body">
        <!-- <p>Some text in the Modal Body</p> -->
        <table>
            <tr>
              <th>Paramter</th>
              <th>Description</th>
              <th>Status</th>
            </tr>
            <tr>
                <td>Parameter1</td>
            </tr>
        </table>
      </div>
      <div class="modal-footer">
        <!-- <h3>M</h3> -->
      </div>
    </div>

</div>

<div class="heading">
    <button id="proceed"><span><h2>Proceed to url</h2></span></button>
</div>

<script>
    // Get the modal
    var modal = document.getElementById("myModal");

    // Get the button that opens the modal
    var btn = document.getElementById("phish_param");

    // Get the <span> element that closes the modal
    var span = document.getElementsByClassName("close")[0];

    // When the user clicks the button, open the modal
    btn.onclick = function() {
      modal.style.display = "block";
    }

    // When the user clicks on <span> (x), close the modal
    span.onclick = function() {
      modal.style.display = "none";
    }

    // When the user clicks anywhere outside of the modal, close it
    window.onclick = function(event) {
      if (event.target == modal) {
        modal.style.display = "none";
      }
    }
    </script>

</body>
</html>
```