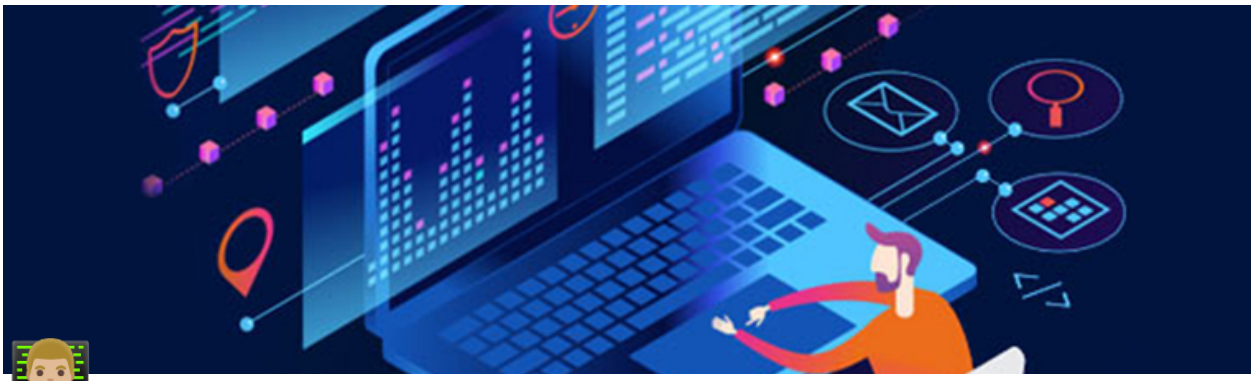

LibreVAPT

A project by Nidhi and Nomaan

Vulnerability Detection using Machine Learning

Introduction

Vulnerability detection can be a complex task and often requires a deep understanding of web applications and their underlying technologies.

To detect vulnerabilities using Python, you can write a script that makes requests to a target website and looks for certain signs of vulnerabilities. For example, to detect SQL injection vulnerabilities, you could write a script that sends different payloads to the target website and looks for changes in the website's behavior or output that would indicate a vulnerability.

Here's a project of a basic script that uses the requests library to send a payload to a target URL and check the response for a specific string that could indicate a vulnerability:

```
import requests

target_url = "http://www.example.com/search?q=query"
payload = "'; SELECT * FROM users --"

response = requests.get(target_url + payload)
if "error" in response.text:
    print("Vulnerability detected!")
else:
    print("No vulnerability detected.")
```

In this example, the payload "'; SELECT * FROM users --" is appended to the target URL and a GET request is sent. If the response contains the string "error", the script prints a message indicating that a vulnerability has been detected.

Here is a simple Python script to detect XSS (Cross-Site Scripting) vulnerabilities in a web application:

```
import requests

def test_xss(target_url, payload):
    response = requests.get(target_url + payload)
    if payload in response.text:
        print(f"[+] XSS vulnerability detected in URL: {target_url}")
    else:
        print(f"[-] No XSS vulnerability detected in URL: {target_url}")
```

```
target_url = "http://www.example.com/search?q="
payloads = [
    "<script>alert(1)</script>",
    "<img src='x' onerror='alert(1)'>",
    "<svg/onload=alert(1)>"
]

for payload in payloads:
    test_xss(target_url, payload)
```

In this example, the function test_xss() takes a target URL and a payload as input makes a GET request to the URL with the payload appended to it and checks the response text for the presence of the payload. If the payload is present in the response text, it means the website is vulnerable to XSS attacks.

About Project

This is an open-source project for vulnerability detection of source code-level based on machine learning techniques.

This project contains a framework which encapsulates 6 mainstream neural network models and can be easily extended to use other network models implemented using Keras.

The framework does not require any code analysis. It takes source code (i.e., functions or files) as input and the output is the probability of the corresponding input sample being vulnerable or not.

For this project, we also collected vulnerable functions from 9 open-source software projects (written in C programming language)

Open Source Projects	Web Page
Asterisk	https://www.asterisk.org/
FFMPEG	https://ffmpeg.org/
HTTPD	https://httpd.apache.org/
LibPNG	https://www.libpng.org/pub/png/libpng.html
LibTiff	https://www.libtiff.org/
OpenSSL	https://www.openssl.org/
Pidgin	https://pidgin.im/
VLC Media Player	https://www.videolan.org/vlc/index.html
Xen	https://xenproject.org/

Required Packages

💡 Anaconda 3
Python Framework and libraries

💡 tensorflow-gpu
Model training & learning (tested on Tensorflow 1.14 and 1.15, not compatible with 2.x)

💡 keras
A high-level abstraction on top of Tensorflow

💡 sklearn
A machine learning toolkit in Python

💡 numpy
Support for arrays & matrices manipulation

💡 pandas
Support for data manipulation and analysis

💡 matplotlib
Result visualization

💡 yaml
Configuration of parser

💡 gensim
A free Python library for topic modelling and natural language processing (supporting Python implementation of Word2vec and FastText)

💡 glove
A Python library for GLoVe implementation

Methodology

Step 1 - Train a Word2vec model

> To use the provided data samples for model training, we have to train a Word2vec model first. By executing the following command:
>
> ```
> python Word_to_vec_embedding.py --data_dir <path_to_code_base> --output_dir <path_to_the_output_file>
> ```
>
> We can train a Word2vec model based on the code specified in the <path to the code base.>. The purpose of training the Word2vec model is to convert source code tokens to meaningful embeddings for the neural network models to learn from.

Step 2 - Train a neural network model

> When the Word2vec model is ready. One can train a neural network model. The parameters related to experiment/model settings are stored in a yaml configuration file. This allows users to conveniently adjust the settings by just changing the configuration file.
>
> Once the configuration file is ready, one may run the following command to train a neural network model.
>
> ```
> python main.py --config config/config.yaml --data_dir <path_to_your_code>
> ```

Step 3 - Test a trained neural network model

> When training is completed, a user can test a network model on the test set by using following command:
>
> ```
> python main.py --config config/config.yaml --test --trained_model <path_of_the_trained_model>
> ```