VulScrape

Vulnerability Detection & Prediction for Forecasting Cyber Attacks

Final Report of the Project

Supervised by

Dr. Ahmedul Kabir

Assistant Professor

Institute of Information Technology,

University of Dhaka

Submitted by

Md. Saleh Ibtasham (Niloy)

Exam Roll:0905

BSSE Session: 2016-2017

Institute of Information Technology,

University of Dhaka

Date of Submission: April 5th, 2021

LETTER OF TRANSMITTAL

4th April, 2021
The Coordinator
Software Project Lab 3
Institute of Information Technology
University of Dhaka

Subject: Submission of the final report of Software Project Lab 3.

Dear Sir,
With due respect, I am pleased to submit the final report on VulScrape, A web based vulnerability detection and prediction tool. Although this report may have shortcomings, I have tried my level best to produce an acceptable report. I would be highly obliged if you overlooked the mistakes and accepted the effort that has been put in this report.

Sincerely yours,
Md. Saleh Ibtasham
Roll: BSSE 0912
Exam Roll: 0905
BSSE 9th batch
Institute of Information Technology
University of Dhaka

Acknowledgment

At first, I would like to thank almighty for helping to prepare the final report of this project.

I would like to express my deepest gratitude to all those who provided me the support and encouragement to start this project. Thanks to my supervisor Dr. Ahmedul Kabir, Assistant Professor, Institute of Information Technology, University of Dhaka, whose continuous suggestions and guidance has been invaluable to me.

I am grateful to the Institute of Information Technology for giving me the opportunity to start such a project.

Lastly, I would like to thank my classmates. They have always been helpful and provided valuable insights from time to time.

Abstract

This document contains the software requirements and specifications, Use-case diagram, Data-based modeling, Class-based modeling, Archetype definition, Mapping requirements to software architecture, Preliminary test plan, High-level description of testing goals, User interface, User Manual, Summary of items and features to be tested in “VulScrape: Vulnerability Detection & Prediction for Forecasting Cyber Attacks”. This is a web based tool. This Tool can be used on C/C++ source codes to find the vulnerabilities present in the code with identifications of those vulnerabilities. The vulnerabilities can be checked for any future exploitations.

Table of Contents

Table of Figures

Index of Tables

Chapter 1: Introduction

1.1 Preamble

SOFTWARE vulnerabilities (or vulnerabilities for short) are a fundamental reason for the prevalence of cyber-attacks. Despite academic and industrial efforts at improving software quality, vulnerabilities remain a big problem. This can be justified by the fact that each year, many vulnerabilities are reported in the Common Vulnerabilities and Exposures (CVE)

In recent years, the number of vulnerabilities discovered and publicly disclosed has shown a sharp upward trend. However, the value of exploitation of vulnerabilities varies for attackers, considering that only a small fraction of vulnerabilities are exploited. Therefore, the realization of quick exclusion of the non-exploitable vulnerabilities and optimal patch prioritization on limited resources has become imperative for organizations. Recent works using machine learning techniques predict exploited vulnerabilities by extracting features from open-source intelligence (OSINT). However, in the face of explosive growth of vulnerability information, there is room for improvement in the application of past methods to multiple threat intelligence. A more general method is needed to deal with various threat intelligence sources. Moreover, in previous methods, traditional text processing methods were used to deal with vulnerability related descriptions, which only grasped the static statistical characteristics but ignored the context and the meaning of the words of the text.

1.2 Motivation

Internet infrastructure plays a crucial role in a number of daily activities. The pervasive nature of cyber systems ensures far-reaching consequences of cyber-attacks. Cyber-attacks threaten physical, economic, social, and political security. The effects of cyber-attacks can disrupt, deny, and even disable the operation of critical infrastructure including power grids, communication networks, hospitals, financial institutions, and defense and military systems. Despite the fact that cyberattacks are constantly growing in complexity, the research community still lacks effective tools to easily monitor and understand them. So if there was a tool which could forecast cyber-attacks it would come in handy to take preventive measures against these attacks before-hand which could save millions of worth data and money.

Vulnerabilities in software are the fundamental cause for cyber attacks. So detecting vulnerabilities from source code and predicting whether that vulnerability may be the cause of exploit in the future is a must. So a tool is necessary for detecting and predicting the software vulnerabilities to counter cyber-attacks in the future.

1.3 Scope of the project

The scope of this project is given below:

- The output of this project will be a web extension.
- The tool will only work on C/C++ programs with source code.
- This tool will be developed and tested on Windows Operating System

1.4 Assumption

Users will only use C/C++ programs as source code as input

1.5 Purpose of the Document

The purposes of this document are:

- Identify the requirements that have to be carried out as the part of the project.
- Form the baseline for construction of the proposed system.
- Help to reduce the development effort and reveal misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

1.6 Definitions

1.6.1 Software Vulnerability

In computer security, a vulnerability is a weakness which can be exploited by a threat actor, such as an attacker, to cross privilege boundaries (i.e. perform unauthorized actions) within a computer system. To exploit a vulnerability, an attacker must have at least one applicable tool or technique that can connect to a system weakness. In this frame, vulnerabilities are also known as the attack surface.

A security risk is often incorrectly classified as a vulnerability. The use of vulnerability with the same meaning of risk can lead to confusion. The risk is the potential of a significant impact resulting from the exploit of a vulnerability. Then there are vulnerabilities without risk: for example when the affected asset has no value. A vulnerability with one or more known instances of working and fully implemented attacks is classified as an exploitable vulnerability—a vulnerability for which an exploit exists. The window of vulnerability is the time from when the security hole was introduced or manifested in deployed software, to when access was removed, a security fix was available/deployed, or the attacker was disabled.

1.6.2 Abstract Syntax Tree (AST)

In computer science, an abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code.

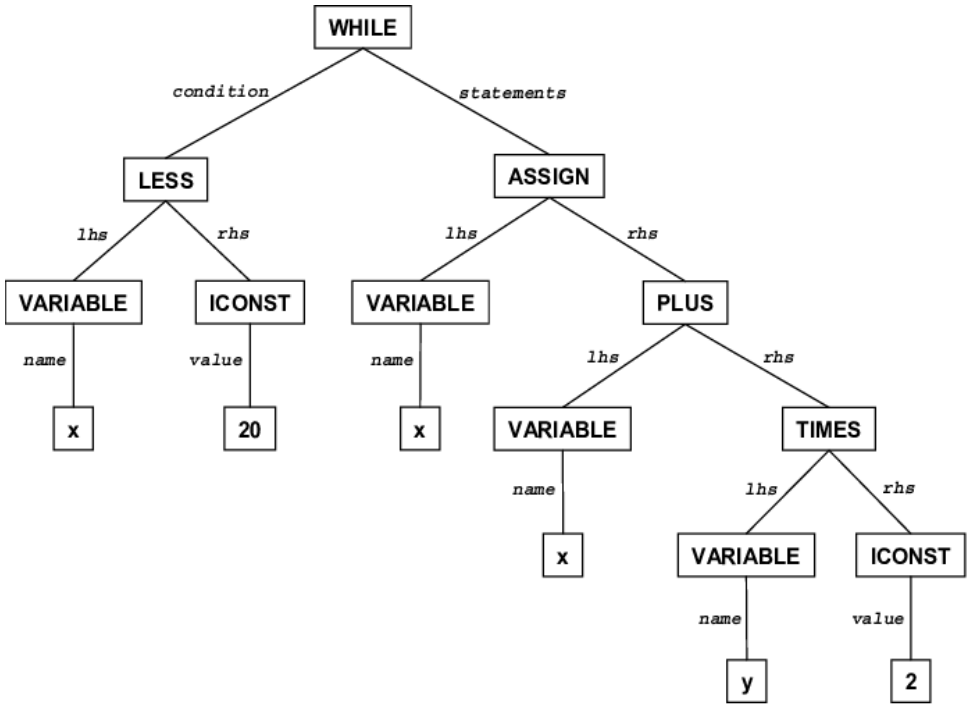


Figure 1 Abstract Syntax Tree (AST)

1.6.3 Control-Flow Graphs (CFG)

In computer science, a control-flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. The control-flow graph is due to Frances E. Allen, who notes that Reese T. Prosser used boolean connectivity matrices for flow analysis before.

In a control-flow graph each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow.

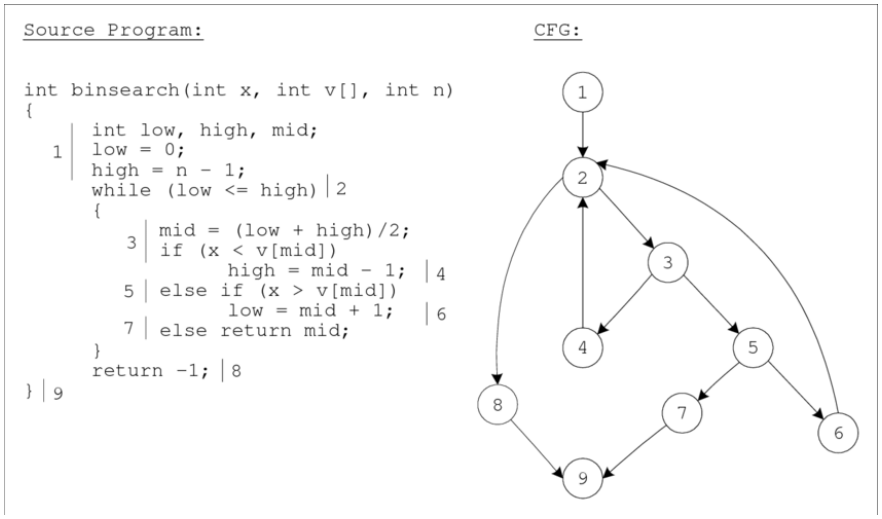


Figure 2 Control-Flow Graphs (CFG)

There are, in most presentations, two specially designated blocks: the entry block, through which control enters into the flow graph, and the exit block, through which all control flow leaves.

1.6.4 Program Dependence Graph (PDG)

In computer science, a program dependence graph is a representation, using graph notation, that makes data dependencies and control dependencies explicit. These dependencies are used during dependence analysis in optimizing compilers to make transformations so that multiple cores are used, and parallelism is improved.

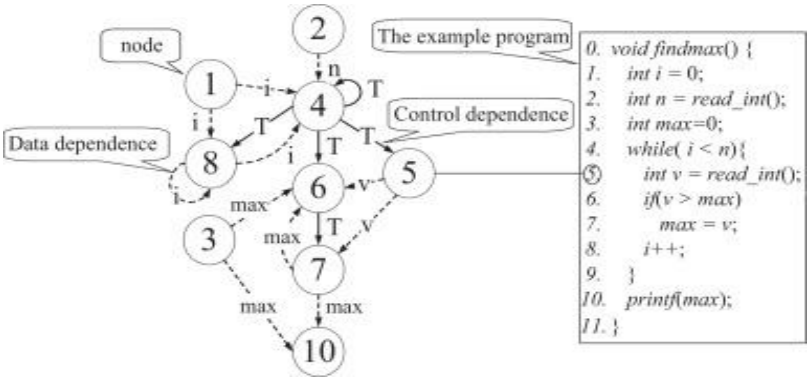


Figure 3 Program Dependence Graph (PDG)

1.6.5 Bidirectional LSTM

Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance on sequence classification problems.

In problems where all timesteps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

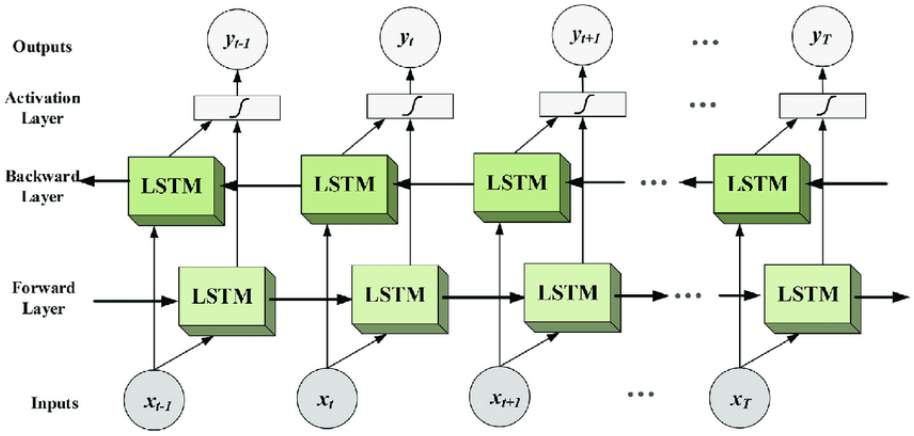


Figure 4 Bidirectional LSTM

1.6.6 FastText

fastText is a library for learning of word embeddings and text classification created by Facebook's AI Research (FAIR) lab The model allows one to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. Facebook makes available pretrained models for 294 languages. fastText uses a neural network for word embedding.

Although it takes longer time to train a FastText model (number of n-grams > number of words), it performs better than Word2Vec and allows rare words to be represented appropriately.

1.6.7 LightGBM

LightGBM, short for Light Gradient Boosting Machine, is a free and open source distributed gradient boosting framework for machine learning originally developed by Microsoft. It is based on decision tree algorithms and used for ranking, classification and other machine learning tasks. The development focus is on performance and scalability.

The LightGBM framework supports different algorithms including GBT, GBDT, GBRT, GBM, MART and RF. LightGBM has many of XGBoost's advantages, including sparse optimization, parallel training, multiple loss functions, regularization, bagging, and early stopping. A major difference between the two lies in the construction of trees. LightGBM does not grow a tree level-wise — row by row — as most other implementations do. Instead it grows trees leaf-wise. It chooses the leaf it believes will yield the largest decrease in loss. Besides, LightGBM does not use the widely-used sorted-based decision tree learning algorithm, which searches the best split point on sorted feature values, as XGBoost or other implementations do. Instead, LightGBM implements a highly optimized histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption.

Chapter 2: Overall Description

2.1 Quality Function Deployment (QFD)

Quality Function Deployment (QFD) is a process and set of tools used to effectively define customer requirements and convert them into detailed engineering specifications and plans to produce the products that fulfill those requirements. With respect to this project the following requirements are identified.

2.2 Normal Requirements

- It is a cyber-attack prediction tool
- It works with source code provided by the user
- Can explain the risk probability of the vulnerability present in the source
- Provide the user with warning log for the vulnerabilities
- Users can load source code anytime for an immediate identification of a threat in the horizon
- Detect the types of vulnerabilities present in the source code of the user

2.3 Expected Requirements

- Risk probability of each vulnerability will be shown
- The log should contain all the vulnerabilities given by the user
- Users can navigate files from the tool to load any source code of C/C++
- Users can update the prediction model with the new data to improve the performance of the tool
- User friendly instructions from the tool for the user to help him process the source code

2.4 Exciting Requirements

- The tool can tell the user the consequence of the vulnerability if it is predicted as an exploit or cyber-attack.
- Make this tool as an extension of Google Chrome
- Retrieve data from the users source code and make assumptions without the user intervening
- The tool will feature an online mode for updating and improving its prediction capabilities
- The tool will generate logs and store them when in online mode from time to time which can be shown to the user at a later time.

2.5 Usage Scenario

User for this system: Developers who use C/C++ as a codebase

Detection: The detection module of this project takes users source code as an input. From the browser the user can upload their C/C++ source code for reviewing the vulnerabilities present in the code. The tool will then use this source files to match the vulnerabilities present in the code and will give a warning and possible list of vulnerabilities present in the code. The user then can download the list of vulnerabilities and use those vulnerabilities for patching his source code.

Prediction: The prediction module of this project takes users vulnerability list or a single vulnerability and predicts whether those vulnerabilities have been exploited or not. From the browser extension the user can select a file with the vulnerabilities present in it or input the vulnerability Id manually to see the prediction the likelihood of exploitability of the vulnerability. The user then can download the results and save them if necessary.

Table 1 Usage Scenario

Usage Scenario	
Detection	1. Take source code of C/C++ as input 2. Suggest vulnerabilities present in the source code as output 3. Download the list of vulnerabilities present in the code
Prediction	1. List of vulnerabilities or single vulnerability id as input 2. Probability of exploitation of the vulnerability as output 3. Download the result and save

Chapter 3 Use Case Diagram

Use Case Diagrams of cyber-attack forecasting tool are given below:

Level 0: Tool for Forecasting Cyber-attacks

Actor: User

Goal in context: The diagram represents the use case of the whole system

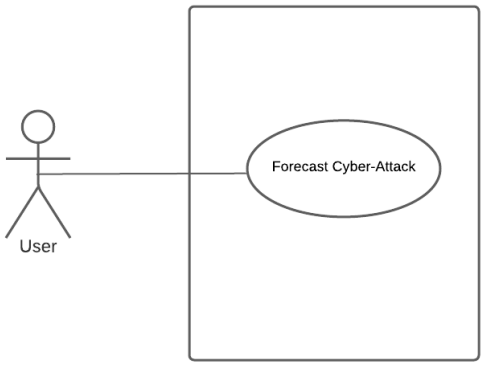


Figure 5 Level 0 Use Case

Level 1: Modules of Tool for Forecasting Cyber-attacks

Actor: User

Goal in context: The diagram represents the use case of all the modules present in the system

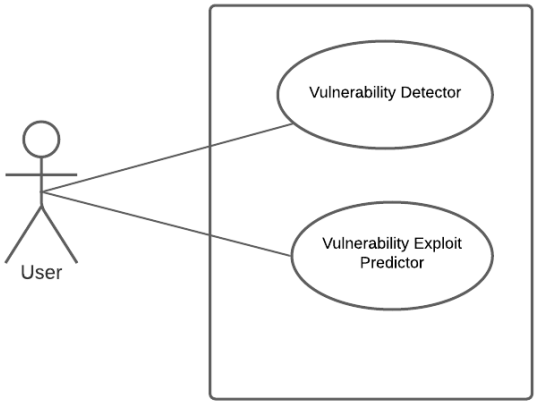


Figure 6 level 1 use case

Level 1.1: Vulnerability Detector

Actor: User

Goal in context: The diagram represents the details of Vulnerability Detector of level 1

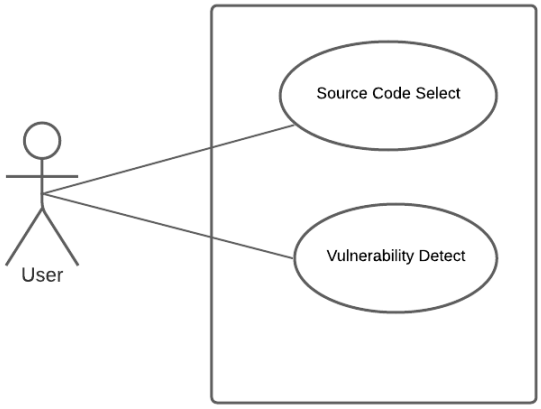


Figure 7 level 1.1 Use Case

Level 1.2: Vulnerability Exploit Predictor

Actor: User

Goal in context: The diagram represents the details of Vulnerability Exploit Predictor of level 1

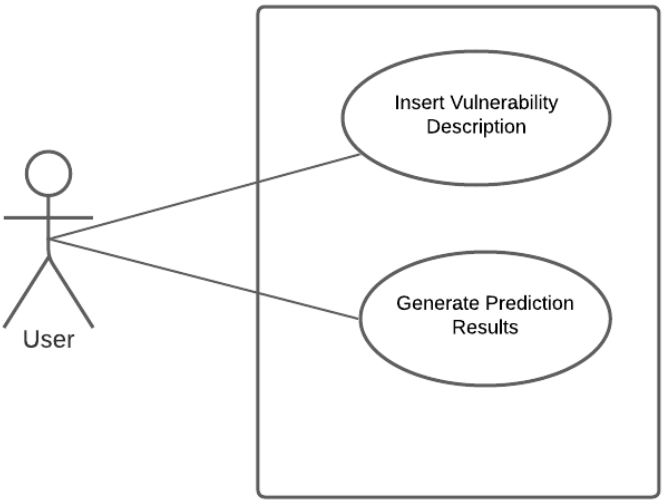


Figure 8 level 1.2 Use Case

Chapter 4: Class Based Model

This chapter describes the class based modeling of the Cyber-attack Forecasting Tool

4.1 Analysis Class

After identifying nouns from scenario, I have filtered nouns belonging to solution domain using General Classification (External entities, Things, Events, Roles, Organizational units, Places, and Structures). Nouns selected as potential class were filtered using Selection Criteria (Retained information, Needed services, Multiple attributes, Common attributes, Common operations, and Essential requirements). After performing analysis on potential classes, I have found the following analysis classes:

Table 2 Analysis Classes

Serial no	Class name	Attributes	Methods
1	Predictor	vulnerabilityId	loadClassifier(), predictVulnerability(), predictionScore(), loadVulnerabilityList()
2	TextProcessor	file	loadFile(), wordStemmer(), textFilter()
3	Embedder	file	loadEmbedder(), loadSentences(), saveEmbedModel(), loadEmbedModel(), generateEmbedModel()
4	Classifier	embedModel, preProcessedFile	setEmbedder(), splitTrainTestSet(), loadClassifier(), trainClassifier(), testClassifier()

5	SourceSlicer	sourceCode	loadSourceCode(), getCfgGraph(), getPdgGraph(), makeLabel(), extractSlices()
6	Graph	indexFile	generateCfgGraph(), generatePdgGraph(), getNodeFromGraph()
7	Mapper	codeSliceFile	loadFunctionCalls(), loadKeywords(), mapKeywordsToCodeSlices()
8	Detector	trainTestFile	loadClassifier(), detectVulnerabilityType(), listCvesOfVulnerabilityType()

4.2 Class Cards

Class: Predictor	
Predicts if the given vulnerability has exploit in the wild	
Responsibility	Collaborator
1. Load the vulnerability for batch prediction 2. Load the classifier 3. Obtain prediction score from the classifier 4. Show predicted results from the classifier	Detector Classifier, Embedder

Class: TextProcessor	
Process input file with listed vulnerabilities with features	
Responsibility	Collaborator
1. Load the vulnerability file 2. Stem textual features 3. Filter textual features 4. Vectorize the sentences	 Embedder

Class: Embedder	
Perform wordEmbedding on the sentences in the input file	
Responsibility	Collaborator
1. Load the vectorized sentences 2. Load wordEmbedding model 3. Save wordEmbedding model 4. Train wordEmbedding model 5. Perform wordEmbedding on test file	TextProcessor Mapper

Class: Classifier	
Perform classification for train and test, validation of models	
Responsibility	Collaborator
1. Load classifier models 2. Split data in training and testing parts 3. Perform training based on hyper-parameters 4. Save trained model 5. Load trained models	 Predictor, Detector Predictor, Detector

Class: SourceSlicer	
Perform code slicing from semantic vulnerability candidates	
Responsibility	Collaborator
1. Load Source Codes 2. Get CFG (Control flow graph) 3. Get PDG (Program Dependency Graph) 4. Extract program slices 5. Label slices with vulnerability types	 Graph Graph Graph Mapper

Class: Graph	
Generate CFG & PDG	
Responsibility	Collaborator
1. Load AST (Abstract Syntax Tree) from source code 2. Generate CFG (Control flow graph) 3. Generate PDG (Program Dependency Graph) 4. Lookup graph nodes	SourceSlicer SourceSlicer SourceSlicer

Class: Mapper	
Map function calls with code slices	
Responsibility	Collaborator
1. Load function calls 2. Load program keywords 3. Load regular expressions for functions and operators 4. Map key function calls with the codeslices 5. Tokenize function calls with sensitive functions 6. Fetch saved function tokens	SourceSlicer Predictor

Class: Detector	
Perform detection of possible vulnerability class from the source code	
Responsibility	Collaborator
1. Load Detection mechanism 2. Detect vulnerability type 3. List possible vulnerability from vulnerability type	Classifier Mapper, SourceSlicer

4.3 Class Diagram

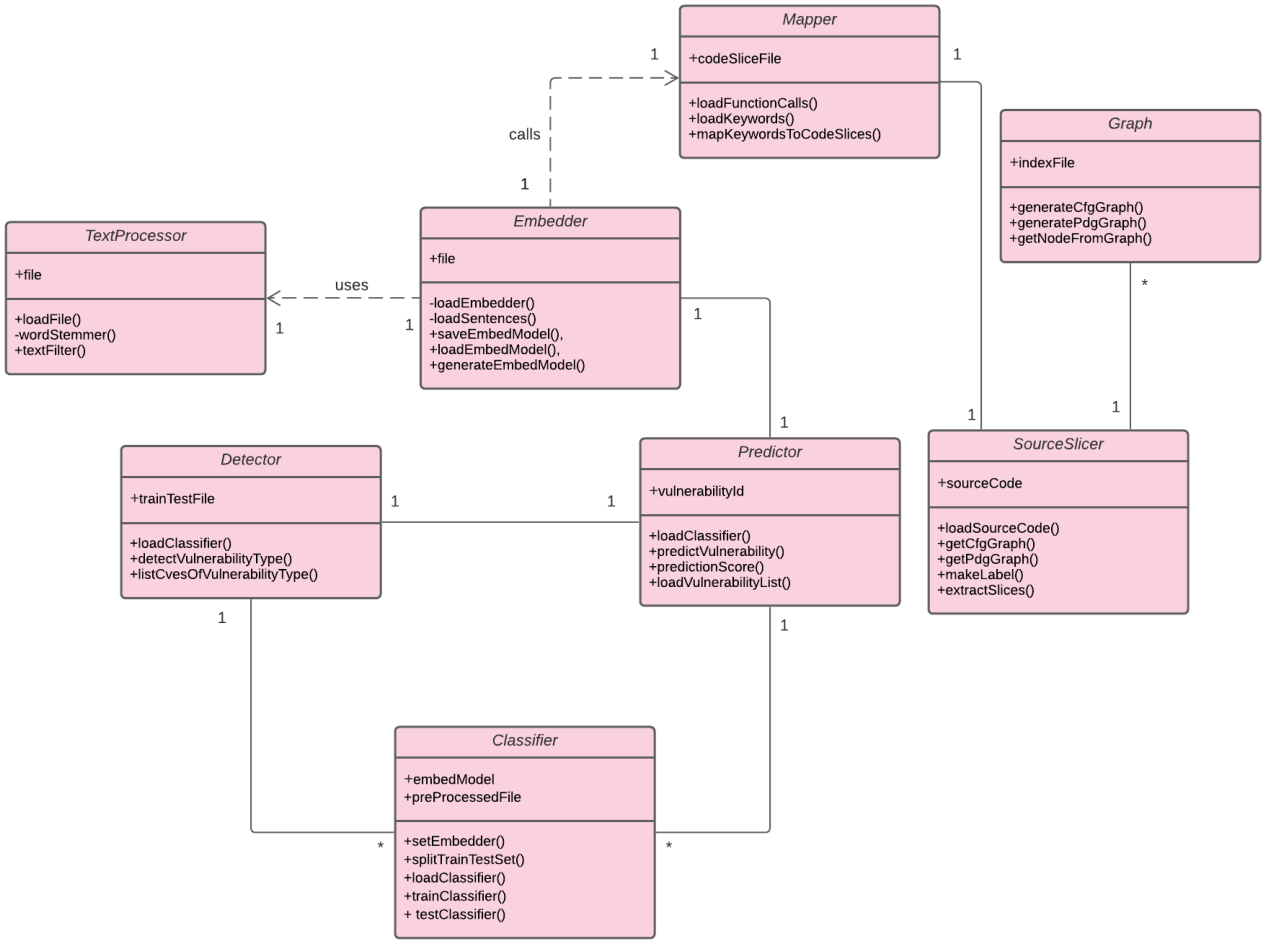


Figure 9 Class Diagram

Chapter 6: Architectural Design

This chapter describes architectural overview and architectural context diagram of the forecasting tool.

6.1 Architectural Overview

This Forecasting tool will follow 2-tier architecture. It will be divided into 2 parts: **presentation layer, logic layer**.

Presentation layer

Figure 11 Web App

Logic layer

Figure 10 Srver

Presentation Layer: The presentation layer is responsible for accepting user input and displaying data to the user. It is a client-side web app. It requests a logic layer to know the vulnerabilities of a software source code and also predict vulnerability exploit.

Logic Layer: The logic layer is a REST API that provides URL endpoints for the presentation layer to communicate with the server. In the system logic layer performs vulnerability detection and exploit prediction.

6.2 Architectural Context Diagram

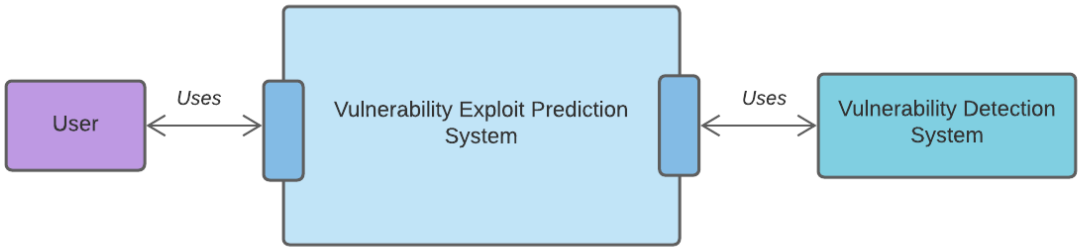


Figure 12 Architectural Context Diagram

Superordinate System: No other system uses the exploit prediction as their part

Subordinate System: The system doesn't use any other system for computation

Peer-level System: The vulnerability Detection system works as a peer with the prediction system

Actors: User is the only actor in the system who uploads source codes and vulnerability input files to the system

Chapter 7: Test Plan

Black box testing technique has been used to test the Forecasting tool. All the tests have been conducted on Windows 10 (64 bit)

7.1 High-level description of testing goals

The Cyber-attack Forecasting tool will undergo high level testing which is popularly known as Black-box testing. Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

Serving as a bridge between users and development team of a product, the ultimate goal of software testing is to troubleshoot all the issues and bugs as well as control the quality of a resulted product. The goals of high level software testing are given below:

- To ensure that the system works properly that means the system can edit pictures correctly
- To ensure that the system satisfies the user requirements and works as desired.
- To find any existing bug
- To improve the system

7.2 Summary of items and features to be tested

Test ID	Test case	Input Data Test	Steps to be Executed	Expected Result	Actual Result	Pass/Fail
T1	Test if system can upload a source code	Source code	1.click upload button 2. search a source code	The source code will be uploaded	The source is uploaded	Pass
T2	Test if system can detect vulnerability	Source Code	1.click detect button	List of Vulnerability will be shown	Vulnerabilities shown in the with a table	Pass
T3	Test if the vulnerability	Source Code	1.click download button	A file with the list of	Vulnerabilities and other log	Pass

	can be downloaded			vulnerability will be downloaded	information are downloaded in JSON format	
T4	Test if system can predict a single vulnerability	Vulnerability Id	1.click predict button	The risk probability score will be shown	CVE-ID is shown from known list of vulnerabilities	Pass
T5	Test if system can predict a batch of vulnerability	file	1.click predict button	Risk probability of the batch of vulnerabilities will be shown	Risk Probabilities with vulnerability confirmation is shown in the table	Pass
T6	Test if vulnerability prediction scores can be downloaded	file	1.click predict button 2. click download button	A file with scores of risk probability of the vulnerabilities	A file with JSON format can be downloaded	Pass
T7	Test if system can update the prediction model	file	1.click update model button	The prediction model will updated with a successfully updated message	The prediction model is not updated using new data	Fail
T8	Test if system can train-test with the new model		1.click train button	The system will return a successful train test message	The system can load and predict with new trained model	Pass

Table 3 Table of individual tests

Chapter 8: Methodology

Here in the tool the whole framework is divided into 2 parts. The parts are comprised of: **Detection** and **Prediction**

The framework used in the app is mentioned as below:

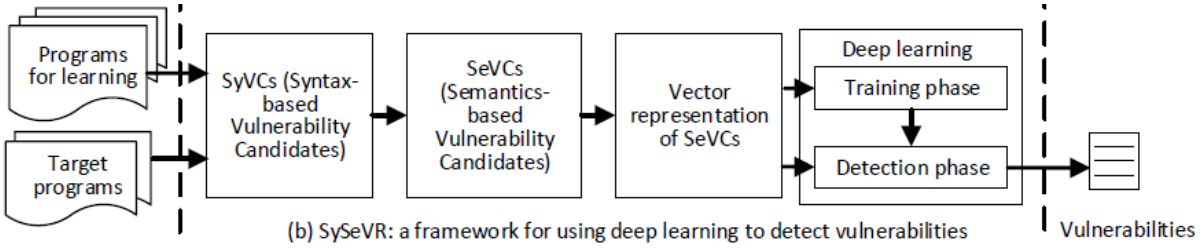


Figure 14 The SySeVR framework is inspired by the notion of region proposal and is centered on obtaining SyVC, SeVC, and vector representations of programs.

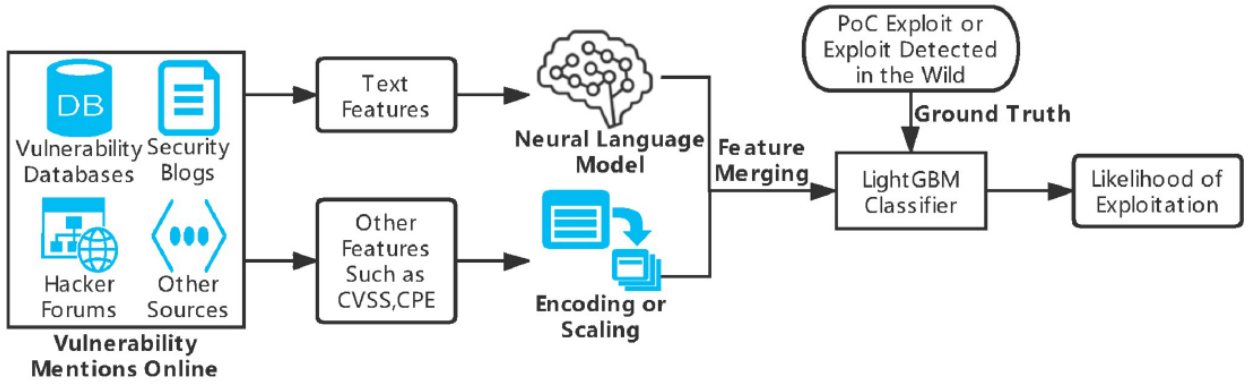


Figure 13 Overview of fastEmbed

8.1 The Detection Module

We observe that vulnerabilities exhibit some syntax characteristics, such as function call or pointer usage. Therefore, we are using syntax characteristics to identify SyVC[1]s, which serve as a starting point for vulnerability detection (i.e., SyVCs are not sufficient for training deep learning models because they accommodate no semantic information of vulnerabilities). Fig. 14 highlights the framework inspired by the notion of region proposal. Essentially, the framework seeks SyVC, SeVC[2], and vector representations of programs that are suitable for vulnerability detection.

In order to help understand **the detection module**, we use the running example described in Fig. 2 to highlight how the framework extracts SyVC, SeVC, and vector representations of programs. At a high level, a SyVC, which is highlighted by a box in Fig. 15, is a code element that matches the syntax characteristics of some vulnerability. A SeVC extends a SyVC to include statements (i.e., lines of code) that are semantically related to the SyVC, where semantic information is induced by control dependency and/or data dependency; this "SyVC to SeVC" (or SyVC!SeVC) transformation is fairly involved and will be elaborated later (in Fig. 3). Finally, each SeVC is encoded into a vector for input to a deep neural network.

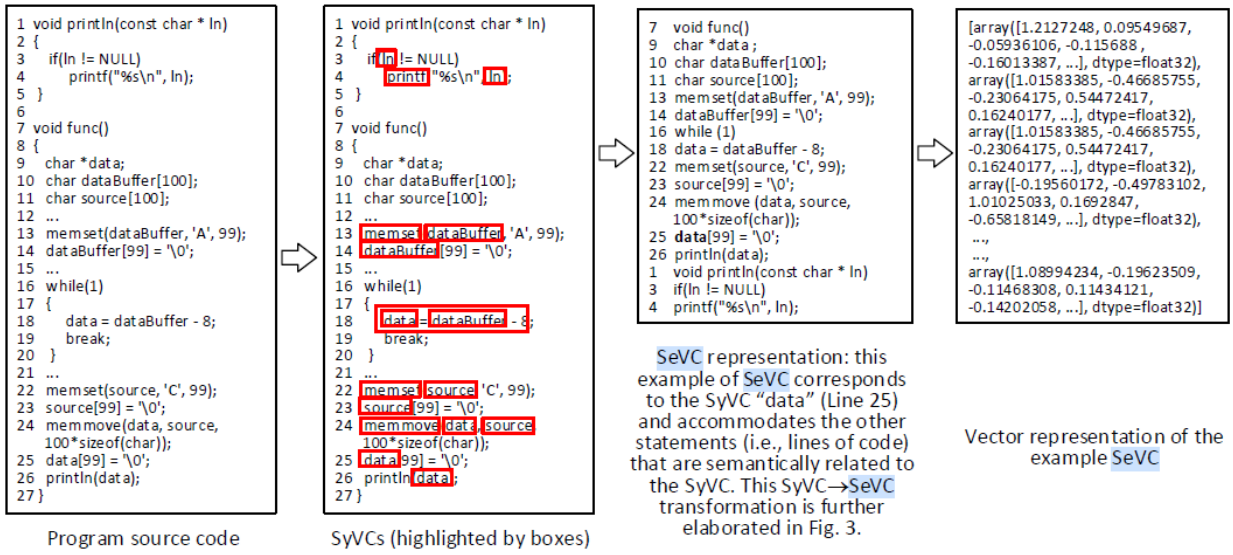


Figure 15 An example illustrating SyVC, SeVC, and vector representations of program, where SyVCs are highlighted by boxes and one SyVC may bepart of another SyVC. The SyVC!SeVC transformation is elaborated in Fig. 16

The below steps are important for developing the detection module.

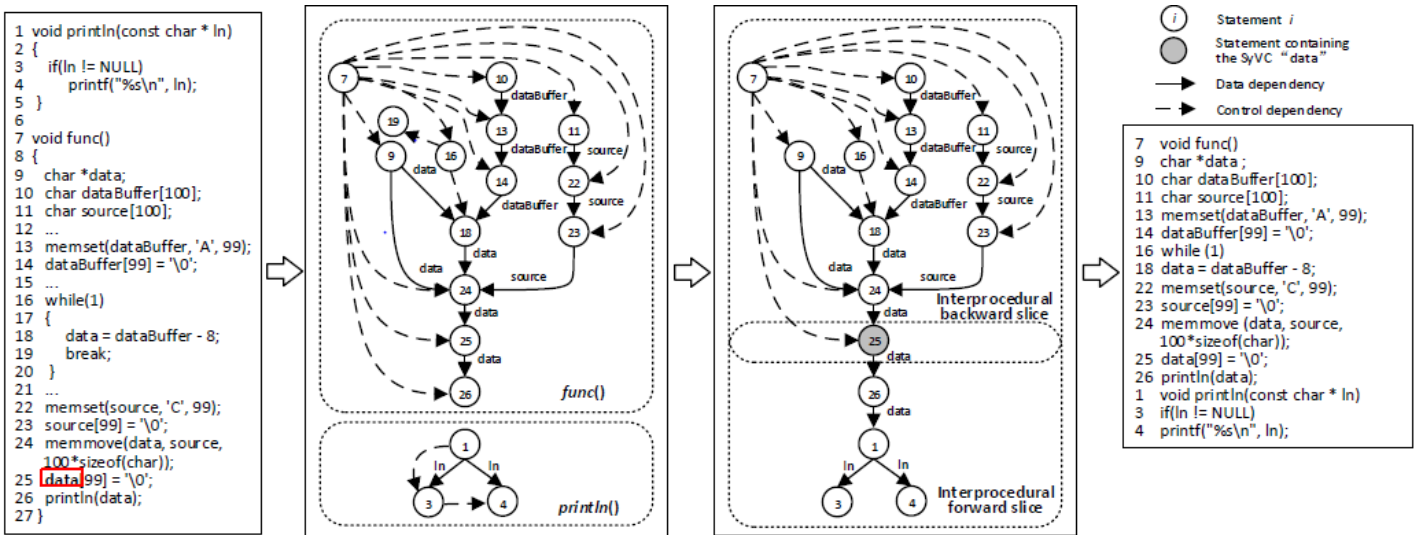


Figure 16 Elaborating the SyVC->SeVC transformation for SyVC "data",

8.1.1 Data Collection

We collected a vulnerability dataset from two sources: NVD (National Vulnerability Database) and SARD (Software Assurance Reference Dataset). NVD contains vulnerabilities in software products (i.e., software systems) and possibly diff files describing the difference between a vulnerable piece of code and its patched version Fig (). SARD contains production, synthetic and academic programs (also known as test cases), which are categorized as "good" (i.e., having no vulnerabilities), "bad" (i.e., having vulnerabilities), and "mixed" (i.e., having vulnerabilities whose patched versions are also available). Note that a program in NVD consists of one or several files (e.g., .c or .cpp files) which contain some vulnerability (corresponding to a CVE ID) or its patched version, and that a program in SARD is a test case.

```
int CVE_2004_1151_VULN_sys32_ni_syscall(int call)
{
    struct task_struct *me = current;
    static char lastcomm[8];
    if (strcmp(lastcomm, me->comm)) {
    printk(KERN_INFO "IA32 syscall %d from %s not implemented\n", call,
           current->comm);
        strcpy(lastcomm, me->comm);
    }
    return -ENOSYS;
}
```

Figure 17 Vulnerable Code Bug

```
int CVE_2004_1151_PATCHED_sys32_ni_syscall(int call)
{
    struct task_struct *me = current;
    static char lastcomm[sizeof(me->comm)];

    if (strncmp(lastcomm, me->comm, sizeof(lastcomm))) {
    printk(KERN_INFO "IA32 syscall %d from %s not implemented\n", call,
    me->comm);
    strncpy(lastcomm, me->comm, sizeof(lastcomm));
    }
    return -ENOSYS;
}
```

Figure 18 Pached Code

8.1.2 Extracting SyVCs

Given that there are many vulnerabilities, we anticipate it to be extremely time-consuming to define and extract their syntax characteristics because this requires to extract the vulnerable lines of code from the vulnerable programs. We use the attributes of nodes on the Abstract Syntax Tree (AST) of a program to describe vulnerability syntax characteristics.

Regardless of the specific descriptions of vulnerability syntax characteristics, there are a set of vulnerability syntax characteristics, where a corresponding code syntax can be matched to have the same characteristics thus getting the vulnerable syntax candidates (SyVC).

Given a function, there are standard routines for generating its AST. The root of the AST corresponds to a function, a leaf of the AST corresponds to a token, and an internal node of the AST corresponds to a statement or multiple consecutive tokens. Intuitively, a SyVC corresponds to a leaf node of an AST, meaning that it is a token, or corresponds to an internal node of an AST, meaning that it is a statement or consists of multiple consecutive tokens. In the second column of Fig. 15, we use boxes to highlight all of the SyVCs that are extracted from the program source code using the vulnerability syntax characteristics from the database.

8.1.3 Transforming SyVCs to SeVCs

In order to detect vulnerabilities, we transform SyVCs to SeVCs (i.e., SyVC → SeVC) to accommodate the statements that are semantically related to the SyVCs in question. For this purpose, we will use the program slicing technique to identify the statements that are semantically related to SyVCs. In order to use the program slicing technique, we need to use Program Dependency Graph (PDG). This requires us to use data dependency and control dependency, which are defined over Control Flow Graph (CFG).

Given PDGs, we can extract program slices from SyVCs. We consider both forward and backward slices because (i) a SyVC may affect some subsequential statements, which may therefore contain a vulnerability; and (ii) the statements affecting a SyVC may render the SyVC vulnerable.

8.1.4 Encoding SeVCs into Vectors

we use word2vec to encode the symbols of the SeVCs (extracted from the 15,591 programs) into fixed-length vectors. The main hyper-parameters include: the dimensionality of word vectors is 30, the window size is 5, the training algorithm is skip-gram, and the threshold for configuring which higher-frequency words are randomly down-sampled is 0.001. Then, each SeVC is represented by the concatenation of the vectors representing its symbols. We set each SeVC to

have 500 symbols (padding or truncating if necessary, as discussed in Algorithm 3) and the length of each symbol is 30, meaning $\Theta = 15; 000$.

8.1.5 Generating Ground-truth Labels of SeVCs

We generate ground-truth labels for the SeVCs in two steps. First, we generate preliminary labels automatically. For SeVCs extracted from NVD, we examine the vulnerabilities whose diff files contain line deletion, while noting that we do not consider the diff files that only contain line addition because NVD does not give the vulnerable statements in such cases. For a diff file containing line deletion, we parse it to mark and distinguish (i) the lines (i.e., statements) that are prefixed with "-" and are deleted/modified from (ii) the lines that are prefixed with "-" and are moved (i.e., deleted at one place and added at another place). If a SeVC contains at least one deleted/modified statement that is prefixed with "-", it is labeled as "1" (i.e., vulnerable); if a SeVC contains at least one moved statement prefixed with "-" and the detected file contains a known vulnerability, it is labeled as "1"; otherwise, it is labeled as "0" (i.e., not vulnerable).

For SeVCs extracted from SARD, a SeVC extracted from a "good" program is labeled as "0" (i.e., not vulnerable); a SeVC extracted from a "bad" or "mixed" program is labeled as "1" (i.e., vulnerable) if the SeVC contains at least one vulnerable statement; otherwise, it is labelled as "0".

8.1.6 Detection

The use BLSTM (Bidirectional Long Short-Term Memory) and the SeVCs accommodating semantic information induced by data and control dependencies. We randomly choose 30,000 SeVCs extracted from the training programs as the training set and 7,500 SeVCs extracted from the test programs as the test set. Both sets contain SeVCs corresponding to the 4 kinds of SyVCs, proportional to the ratio of vulnerable vs. non-vulnerable SeVCs in each kind of SyVCs

8.2 The Prediction Module

The prediction module predicts whether a vulnerability published in vulnerability databases will have PoC exploits or be exploited in the wild. Our vulnerability prediction model consists of two components: **feature merging** and **vulnerability classification**.

8.2.1 Data Collection

The features fed into the prediction module are extracted from open source intelligence data such as NVD. The label of the prediction model is determined by indicating whether there exists corresponding PoC or whether the exploitation of a vulnerability is detected.

The ground truth of the vulnerabilities being exploited are collected form Exploit-DB and PoC exploits.

ID	E	publishedDate	lastModifiedDate	AV	AC	A	CI	II	AI	BS	S	ES	IS	OAP	OUP	OOP	UIR	VEN	DESC
CVE-2013-0001	0	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	PARTIAL	NONE	NONE	4.3	MEDIUM	8.6	2.9	FALSE	FALSE	FALSE	TRUE	1	The Windows Forms (aka WinForms) component in Microsoft .NET Framew
CVE-2013-0002	1	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Buffer overflow in the Windows Forms (aka WinForms) component in Micr
CVE-2013-0003	1	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Buffer overflow in a System.DirectoryServices.Protocols (S.DS.P) namespa
CVE-2013-0004	1	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Microsoft .NET Framework 1.0 SP3, 1.1 SP1, 2.0 SP2, 3.0 SP2, 3.5, 3.5.1, 4, an
CVE-2013-0005	0	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	LOW	NONE	NONE	NONE	COMPLETI	7.8	HIGH	10	6.9	FALSE	FALSE	FALSE	FALSE	1	The WCF Replace function in the Open Data (aka OData) protocol impleme
CVE-2013-0006	1	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Microsoft XML Core Services (aka MSXML) 3.0, 5.0, and 6.0 does not proper
CVE-2013-0007	1	2013-01-09T18:09Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Microsoft XML Core Services (aka MSXML) 4.0, 5.0, and 6.0 does not proper
CVE-2013-0008	1	2013-01-09T18:09Z	2018-10-12T22:03Z	LOCAL	LOW	NONE	COMPLETE	COMPLETE	COMPLETI	7.2	HIGH	3.9	10	FALSE	FALSE	FALSE	FALSE	1	win32k.sys in the kernel-mode drivers in Microsoft Windows Vista SP2, Wi
CVE-2013-0009	0	2013-01-09T18:09Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	NONE	PARTIAL	NONE	4.3	MEDIUM	8.6	2.9	FALSE	FALSE	FALSE	TRUE	1	Cross-site scripting (XSS) vulnerability in Microsoft System Center Operati
CVE-2013-0010	0	2013-01-09T18:09Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	NONE	PARTIAL	NONE	4.3	MEDIUM	8.6	2.9	FALSE	FALSE	FALSE	TRUE	1	Cross-site scripting (XSS) vulnerability in Microsoft System Center Operati
CVE-2013-0011	1	2013-01-09T18:09Z	2018-10-12T22:03Z	NETWORK	LOW	NONE	COMPLETE	COMPLETE	COMPLETI	10	HIGH	10	10	FALSE	FALSE	FALSE	FALSE	1	The Print Spooler in Microsoft Windows Server 2008 R2 and R2 SP1 and Wir
CVE-2013-0012	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0013	0	2013-01-09T18:09Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	PARTIAL	PARTIAL	NONE	5.8	MEDIUM	8.6	4.9	FALSE	FALSE	FALSE	FALSE	1	The SSL provider component in Microsoft Windows Vista SP2, Windows Se
CVE-2013-0014	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0015	1	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	PARTIAL	NONE	NONE	4.3	MEDIUM	8.6	2.9	FALSE	FALSE	FALSE	TRUE	1	Microsoft Internet Explorer 6 through 9 does not properly perform auto-se
CVE-2013-0016	1	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0017	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0018	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 6 through 9 allo
CVE-2013-0019	1	2013-02-13T12:04Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 7 through 10 all
CVE-2013-0020	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 9 allows remote
CVE-2013-0021	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 6 through 10 all
CVE-2013-0022	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 9 allows remote
CVE-2013-0023	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 9 and 10 allows
CVE-2013-0024	0	2013-02-13T12:04Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 8 and 9 allows re
CVE-2013-0025	1	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 8 allows remote
CVE-2013-0026	1	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 9 allows remote
CVE-2013-0027	1	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 6 through 10 all
CVE-2013-0028	0	2013-02-13T12:04Z	2018-10-12T22:03Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 6 through 9 allo
CVE-2013-0029	1	2013-02-13T12:04Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	TRUE	1	Use-after-free vulnerability in Microsoft Internet Explorer 6 through 9 allo
CVE-2013-0030	0	2013-02-13T12:04Z	2018-10-30T16:27Z	NETWORK	MEDIUM	NONE	COMPLETE	COMPLETE	COMPLETI	9.3	HIGH	8.6	10	FALSE	FALSE	FALSE	FALSE	1	The Vector Markup Language (VML) implementation in Microsoft Internet
CVE-2013-0031	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0032	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0033	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0034	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0035	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0036	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea
CVE-2013-0037	0	2017-05-11T14:29Z	2017-05-11T14:29Z															0	** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Rea

Figure 19 NVD Database

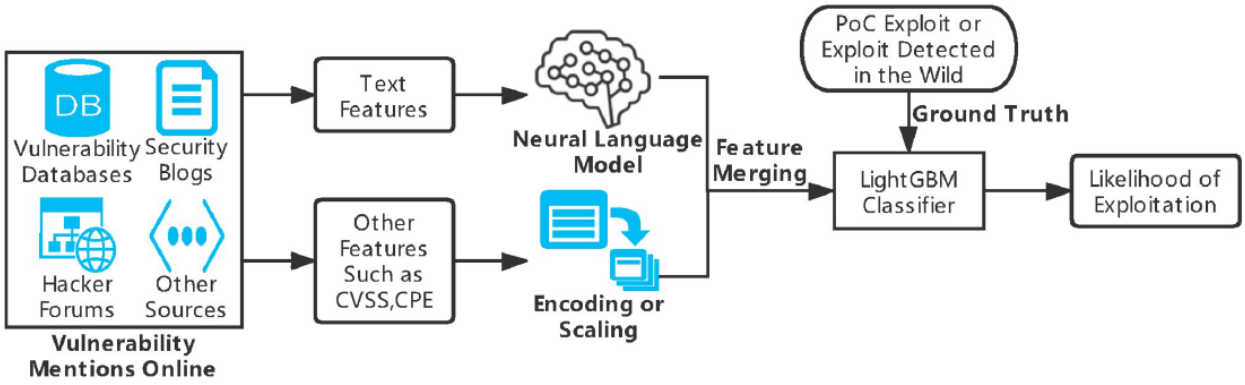


Figure 20 Overview of FastEmbed

8.2.2 Feature Merging

In the feature merging part, the input features of the model are divided into two categories, namely text features and other inherent features of vulnerabilities such as CVSS.

8.2.3 Textual Features Embedding

In terms of text features, through lemmatization, removal of stop words, embedding of the neural language model, we get a more accurate embedded representation of vulnerability related text. Judging from the output results of the text feature training model, through the training of neural

Feature Source	Raw data		Category	Modelled Method
Open Source Vulnerability Data	CVSS	Access Vector	Categorical	One-hot Encoding
		Access Complexity	Categorical	One-hot Encoding
		Authentication	Categorical	One-hot Encoding
		Confidentiality Impact	Categorical	One-hot Encoding
		Integrity Impact	Categorical	One-hot Encoding
		Availability Impact	Categorical	One-hot Encoding
		Base Score	Numeric	Scaled to N(0,1)
		Exploitability Score	Numeric	Scaled to N(0,1)
		ImpactScore	Numeric	Scaled to N(0,1)
	CPE	Application	Binary	Binary Vector
		Hardware	Binary	Binary Vector
		OS	Binary	Binary Vector
		No Product	Binary	Binary Vector
		List	Text	fastText
	CWE	CWE-ID	Numeric	One-hot Encoding
	Product Count		Numeric	Scaled to N(0,1)
	OVAL		Binary	Binary Vector
	Description		Text	fastText

Figure 21 Feature list of the dataset

network model, we can get the embedded representation of the vulnerability- related text or the classification result of whether the vulnerability is exploited from the vulnerability- related text. In the final step of feature

merging, the result of text feature model classification or embedded representation of text features is combined with other robust features as the last feature. Here in the text feature embedding we use FastText embedding model.

The FastText classification model is similar in structure to the CBOW model in word2vec. The difference is that FastText classification model is supervised learning model, and its goal is to average all the word embeddings in the document and get the class label from the hidden layer through a non-linear transformation. FastText improves the performance of basic linear classifiers with fast loss approximation and the key features of rank constraint.

8.2.4 Vulnerability Classification and Prediction

The exploited vulnerabilities are labeled with related PoCs in exploit database while the vulnerabilities exploited in the wild labeled with authoritative attack signatures.

The system uses the LightGBM algorithm as the exploit predictor, which is a fast, distributed, high performance gradient boosting algorithm based on decision tree and performs well on unbalanced data sets. LightGBM fastens the training procedure and achieves higher accuracy by using a histogram-based algorithm and generating more complex trees by following leafwise split approach .

Chapter 9: User Interface Design

9.1 The Detection Module

The graphical interface design is provided in the following part:

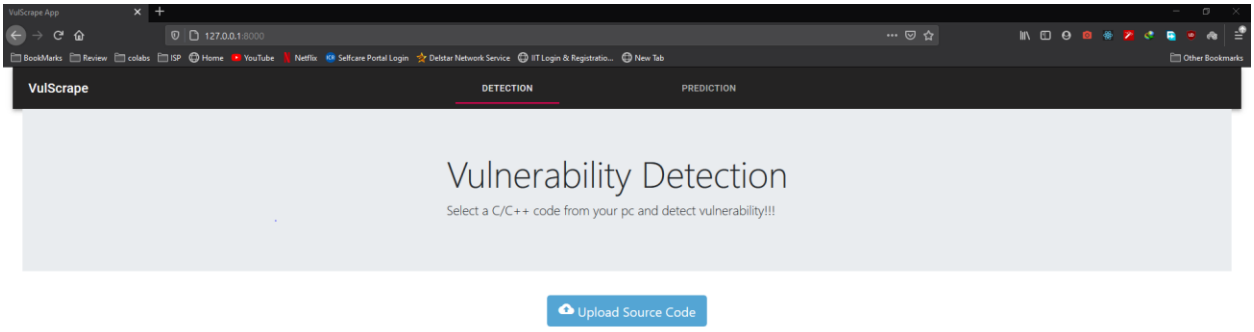


Figure 22: Home page

This is the home page of VulScrape. Here user can upload a specific source code directory present in his computer. He can upload the code using the upload source code button.

After uploading the picture the user will have options mentioned below.

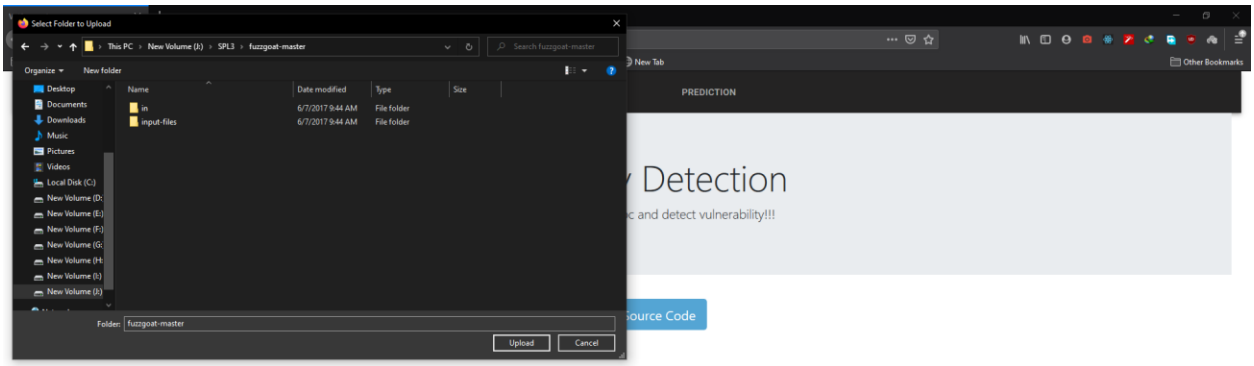


Figure 23: Upload Source Code

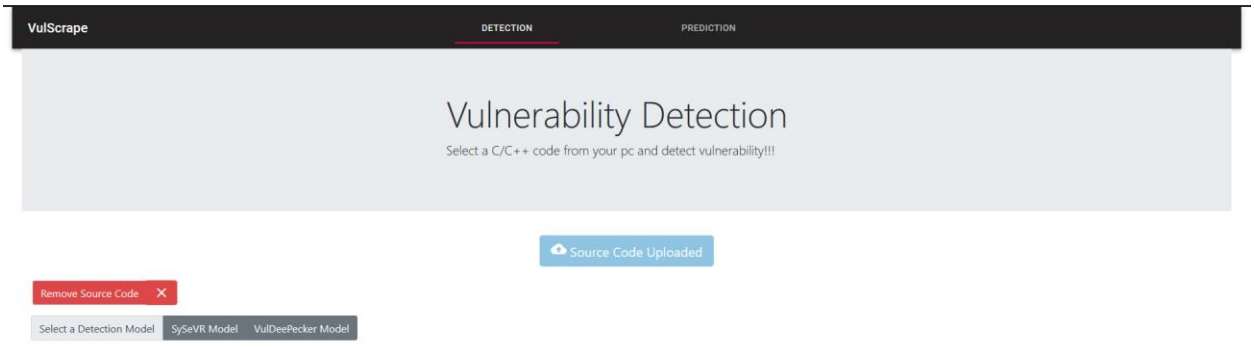


Figure 24: After upload opotions

The user can remove the source code from uploaded selection. The user has 2 choices of models for SySeVR and VulDeePecker model for detection of vulnerability inside code.

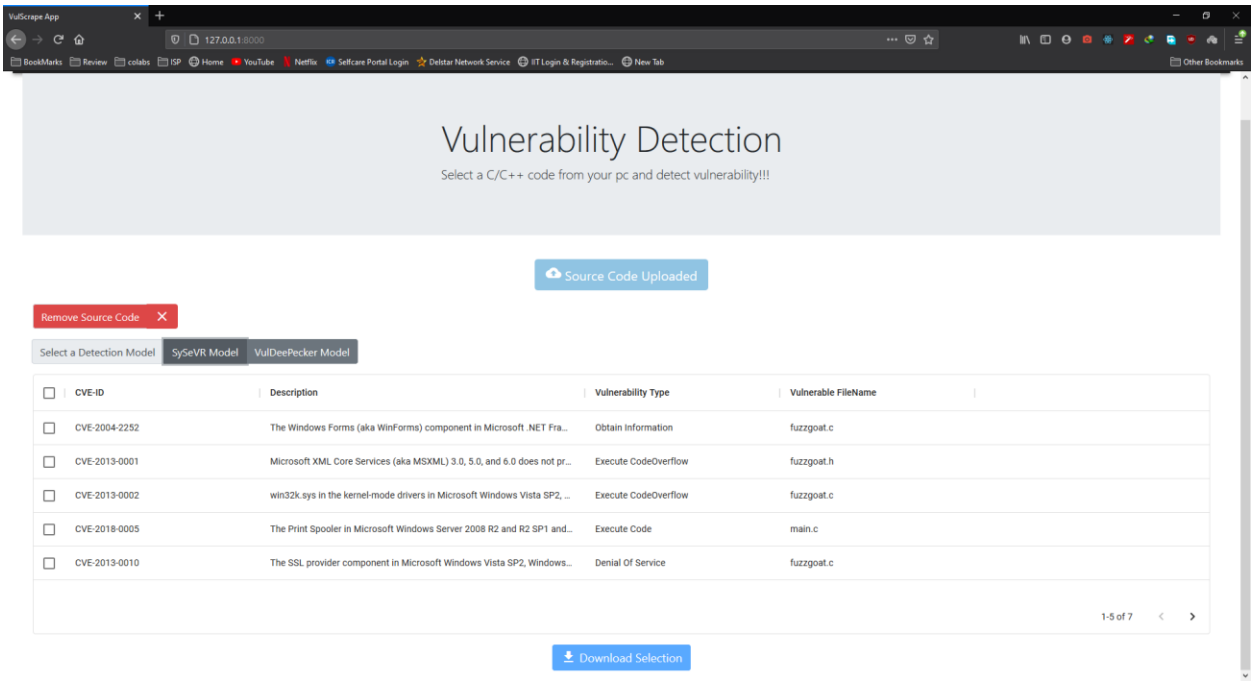


Figure 25: Model Selection Results (1)

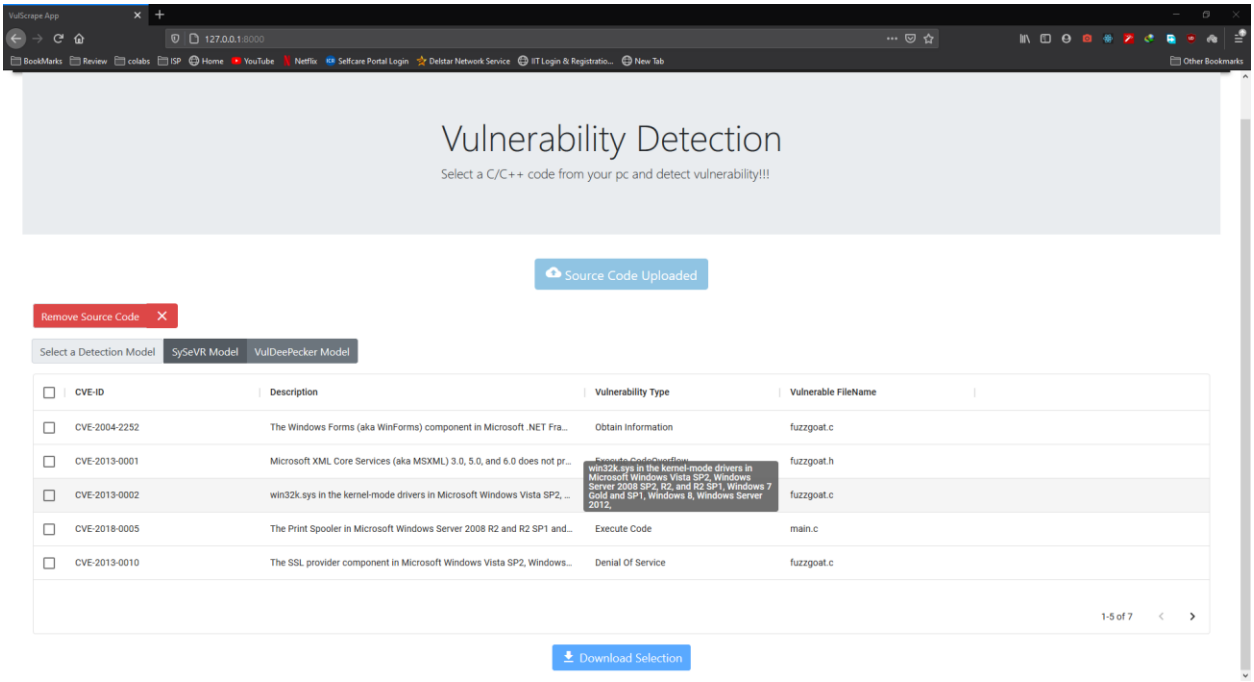


Figure 26: Model Selection Results (2)

The user after selecting a model is shown a grid of results for the detection of code vulnerabilities. The grid list has 4 columns with CVE-ID, Description of that particular CVE, The type of the Vulnerability & The vulnerable code name present in the source directory.

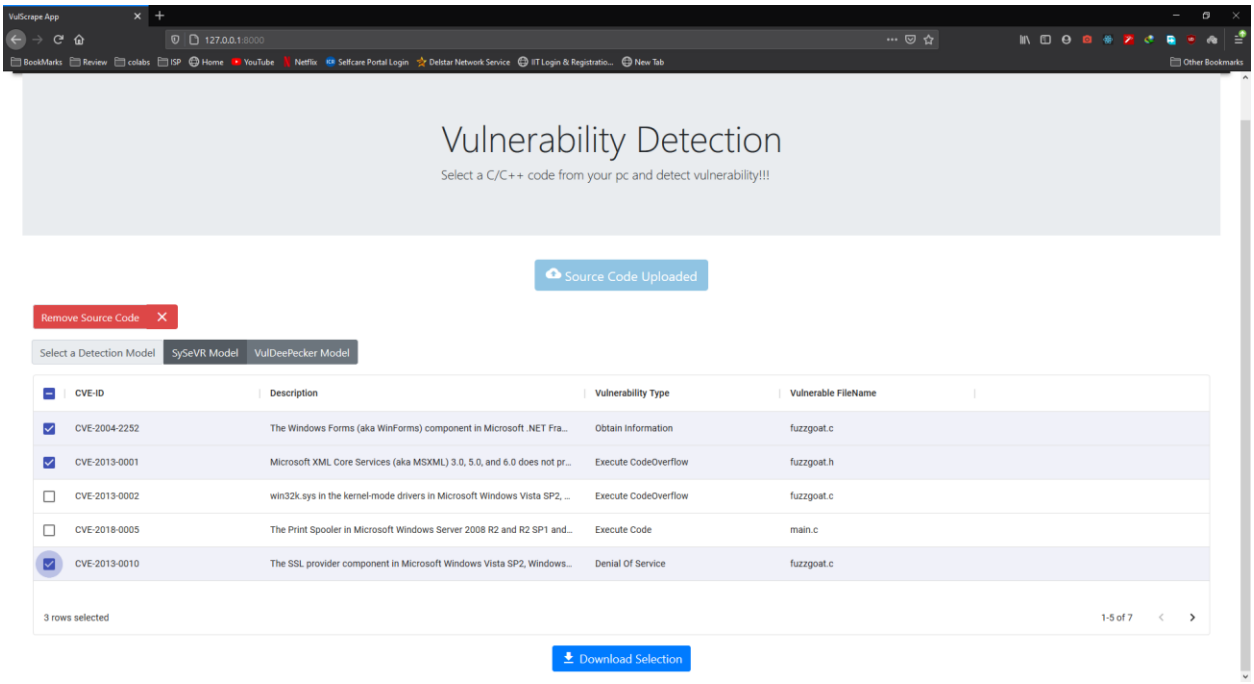


Figure 27: Vulnerability selection

The user can select particular vulnerabilities that he wants to save from the table. The user has a option to download the particular results using the button at the bottom of the page.

After selecting the rows the user can download the particular selections in JSON format using the download option below.

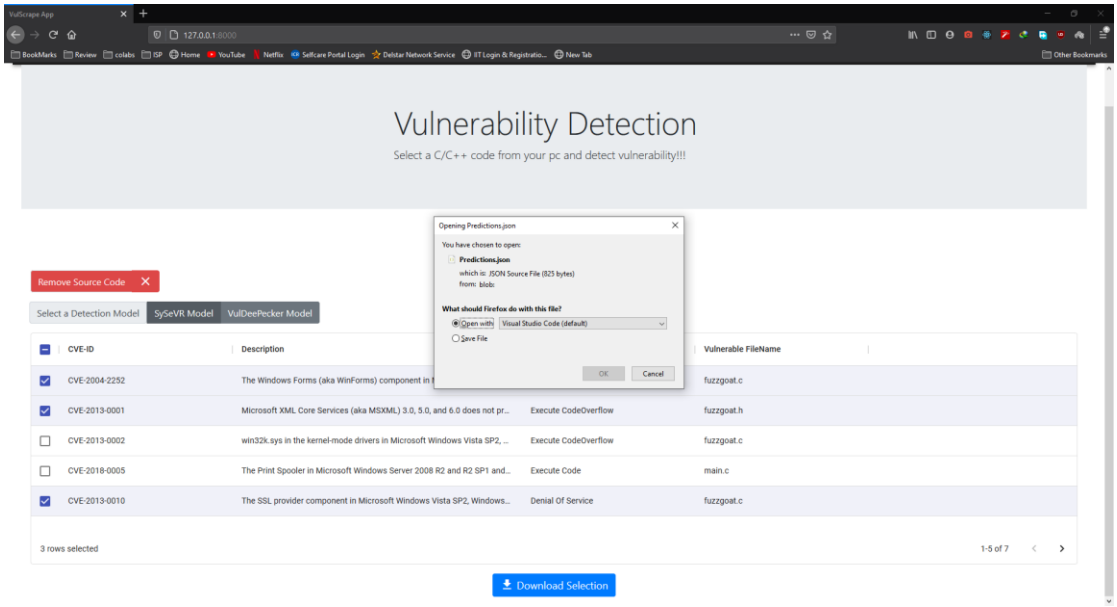


Figure 28: File save optoin

The user can save the file anywhere he wants in his computer.

9.2 The Prediction Module

User can go to the prediction module using the upper navigation bar button called “Prediction.”

The prediction module has a search box where the user can search a CVE with just it’s Id, and the the multiple Ids can be given using comma separated ids.

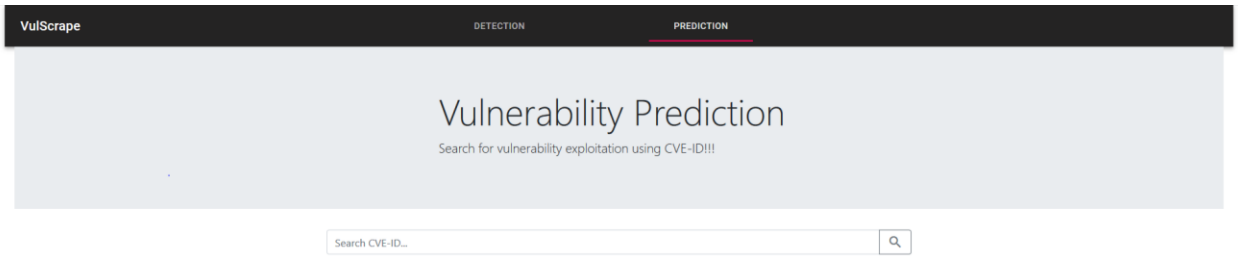


Figure 29: The prediction module

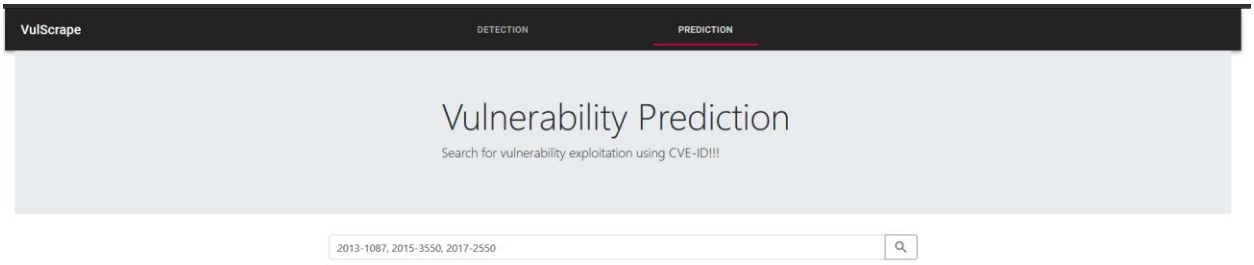


Figure 30: Searching in the box

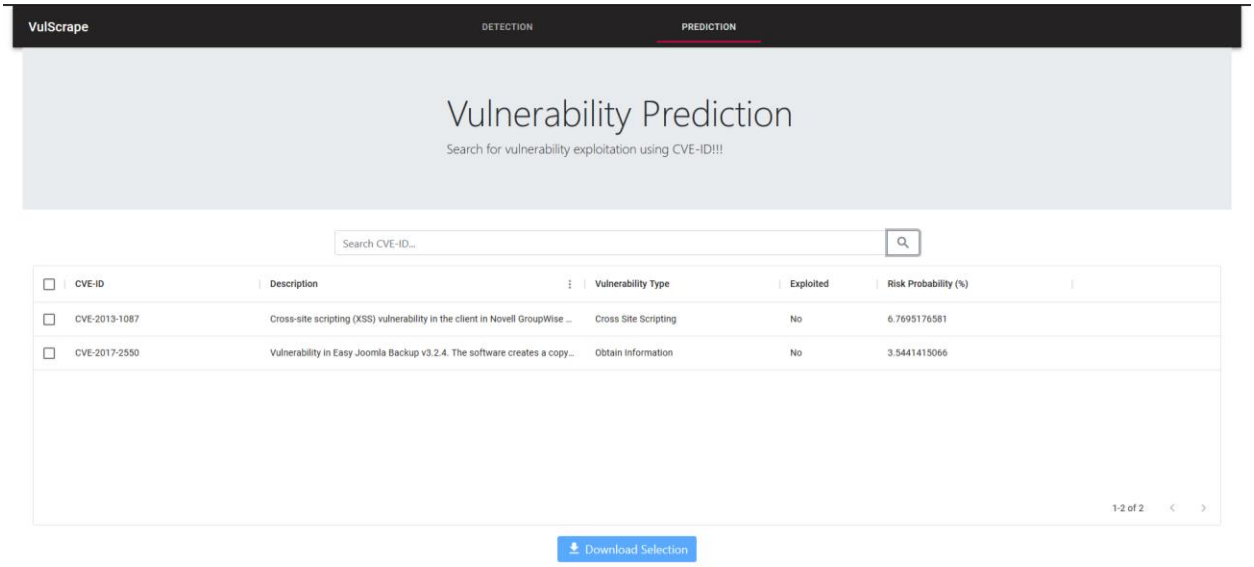


Figure 31: Search result for vulnerabilities

After searching the vulnerabilities the search yields only the legit vulnerabilities present. The vulnerabilities have their descriptions with their exploitation prediction called “Exploited” and “Risk probability” measure.

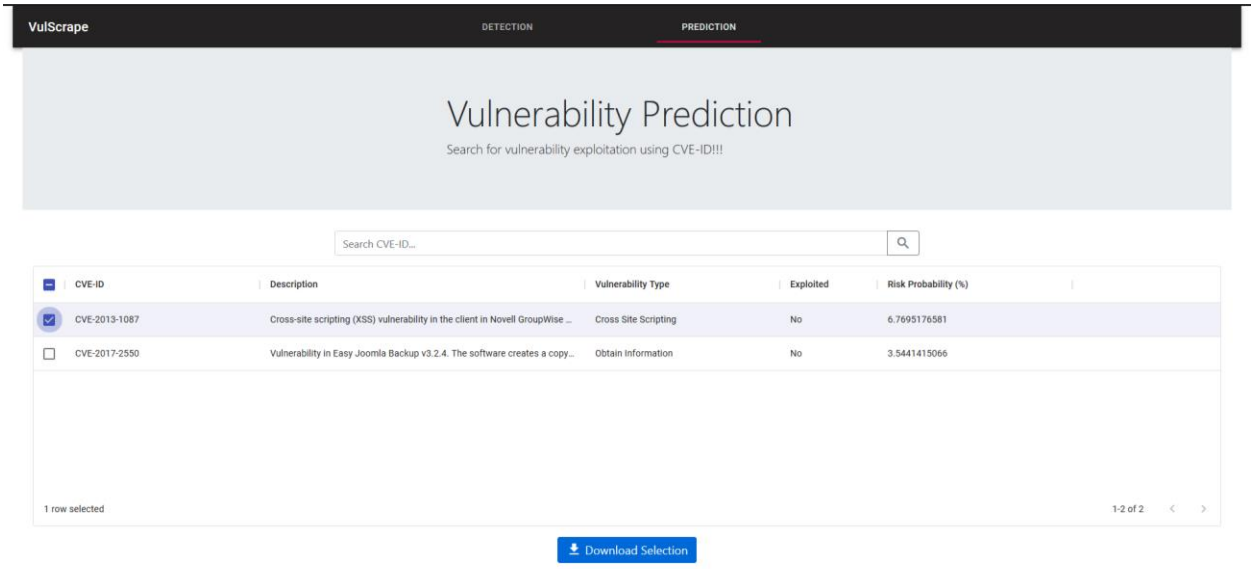


Figure 32: Selection of Prediction results

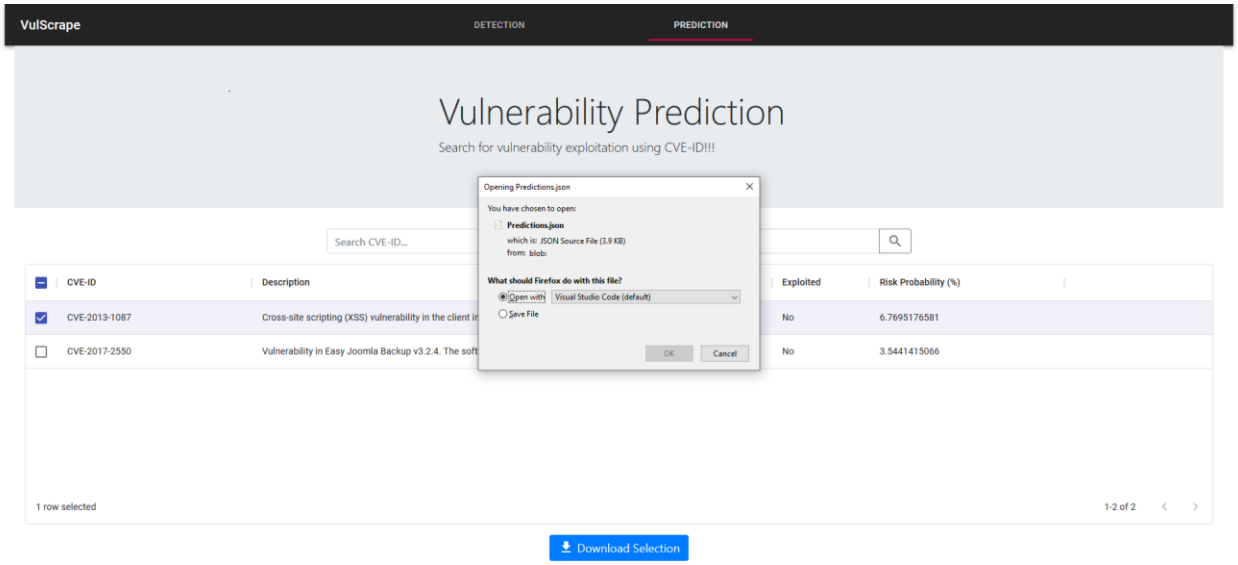


Figure 33: Prediction Reults Download

After the search the users can select and generate JSON reports from the list which has more information about the vulnerabilities.

Chapter 10: Implementation Overview

This chapter aims to describe the implementation process of "VulScrape". The technologies used here are described briefly which were used to build the whole system.

10.1 Technologies Used in Implementation

Static source code analysis is being used all over the world today for code fault detection, defect predictions etc. The technologies that have used in this system implements the most recent technologies and also they are used frequently in code analysis and vulnerability predictions.

- **Python:** Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python supports a lot of modules. Python 3.7.10 have been used here for development

- **Keras:** Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. Kears 2.2.4 have been used in project for building prediction and detection models.

- **Gensim:** Gensim is an open-source library for unsupervised topic modeling and natural language processing, using modern statistical machine learning. Gensim is implemented in Python and Cython. Gensim is designed to handle large text collections using data streaming and incremental online algorithms, which differentiates it from most other machine learning software packages that target only in-memory processing. Gensim 3.8.3 has been used in building the FastText and Word2Vec models for the project.

- **Joern**: Joern is a platform for robust analysis of C/C++ code. It generates code property graphs, a graph representation of code for cross-language code analysis. Code property graphs are stored in a custom graph database. This allows code to be mined using search queries formulated in a Scala-based domain-specific query language. Joern 0.3.1 has been used in the project for code parsing.

- **Neo4j-Community:** Neo4j (Network Exploration and Optimization 4 Java) is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing. Neo4j is implemented in Java and accessible from software written in other languages using the Cypher query language through a transactional HTTP endpoint, or through the binary "bolt" protocol. Neo4j-communtiy-2.1.8 has been used in this project.

- **Django:** Django is a Python-based free and open-source web framework that follows the model-template-views (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an American independent organization established as non-profit. Django 3.1.4 has been used for the full backend of the project.

- **Django Rest Framework:** Django REST framework is a powerful and flexible toolkit for building Web APIs. REST is a loosely defined protocol for listing, creating, changing, and deleting data on your server over HTTP. The Django REST framework (DRF) is a toolkit built on top of the Django web framework that reduces the amount of code you need to write to create REST interfaces. Django Rest 3.12.2 has been used in the project.

- **React:** React (also known as React.js or ReactJS) is an open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality. React have used for the front-end of this project. React 17.0.2 has been used in this project for building the front-end of the application.

10.2 Source Code Description

The classes from CRC are used in the source code for the development.

10.2.1 Predictor

Attributes: cve_list, X_test

Methods	Description
uploadCve()	Gets the cve list from the client side
getCveList()	Fetches cve feature list from available data
get_result()	Perform predictions on the fetched cve lists using their features

10.2.2 Textprocessor

Attributes: working_directory, cfg_db_directory, pdg_db_directory

Methods	Description
deliverCfg()	Gets the control flow graph from source code
deliverPdg()	Gets the program dependency graphs from the source code
deliverCallGraph()	Gets call graph to functions relations
deliverProgramSlices()	Gets the program slice gadget files from the AST

10.2.3 Embedder

Attributes: working_directory, program_slice_path, program_vector_path

Methods	Description
deliverCorpus()	Gets the corpus files for making the embeddings
deliverVectors()	Gets the corpus and makes vectors of the program slices.

10.2.4 Source Slicer

Attributes: working_directory, program_slice_path, program_vector_path

Methods	Description
generateCfg()	Generates the cfg graph of the program
generatePdg()	Generates the pdg_graph of the program
saveProgramSlice()	Saves the program slices in a formatted text document
getVectorSlice()	Generate the vectors of the program slice

10.2.5 Mapper

Attributes: working_directory, function_list, sensitive_function_list

Methods	Description
getProgramFunction()	Gets the particular program function list
getSensitiveFunctions()	Load the sensitive functions and match functions with it present in the program

10.2.6 Detector

Attributes: working_directory, cve_list, program_vector_path, program_slice_path, weight_path

Methods	Description
getDetectionSys()	Gets the sysevr detection model for detecting the programs
getDetectionsVd()	Gets the vuldeepecker detection model for the data
build_mode()	Builds detection model from previously saved weights
testrealdata()	Generate vulnerability detection results for the program loaded

Chapter 11: User Manual Design

What is VulScrape?

VulScrape

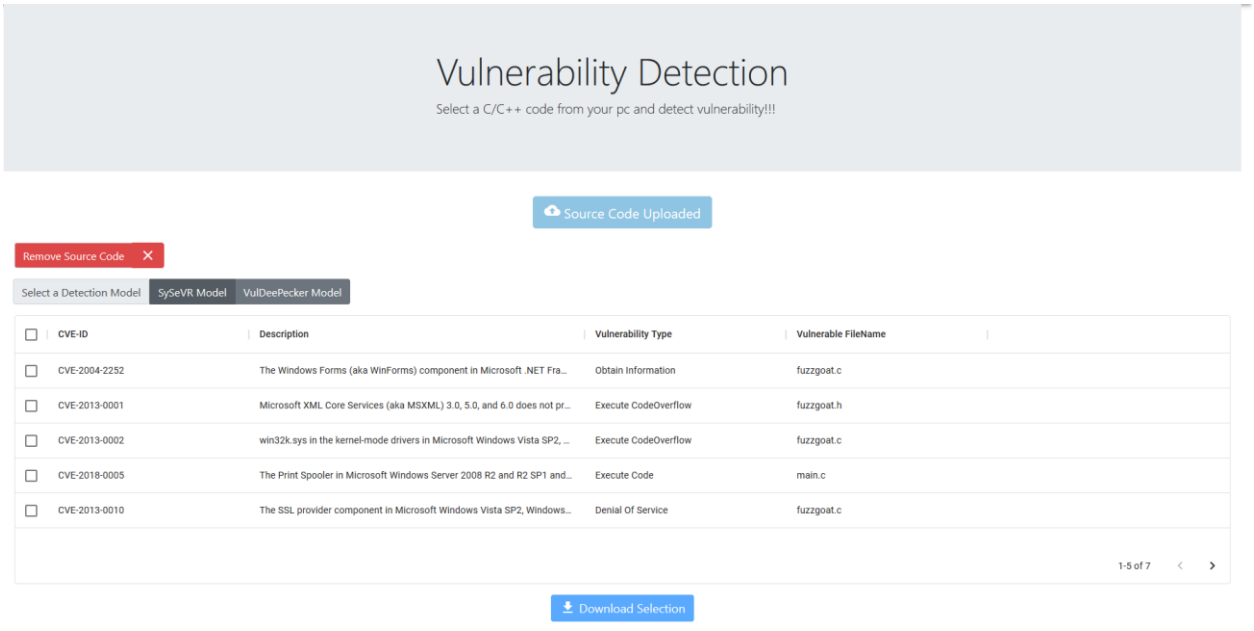
VulScrape is a web based tool for vulnerability detection and prediction for preventing future cyber attacks. We can use this to analyze our C/C++ source code.

What are the features?

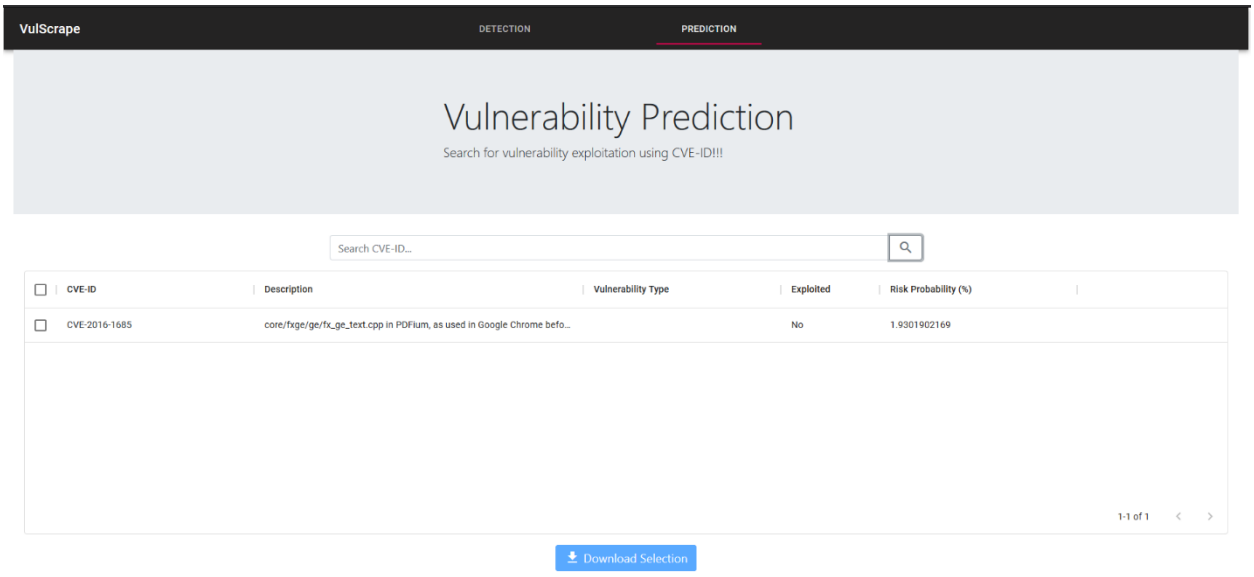
- Detection of source Vulnerability
- Prediction of Vulnerablity

How to use?

1. Copy a source code in the program base directory folder “test_source_code”



2. Select the source code from the web tool using the “Upload Source Code” button
3. The source code will uploaded and 3 options of removing theh source and selecting detection scheme from 2 models, SySeVR and VulDeePecker models will be available
4. Select any one of the options and the list of vulnerabilities present in the code will be shown in a table.



5. The results can be downloaded after selection.
6. The prediction tab has a search bar where the CVE IDs are put.
7. The IDs can be comma separated or single which will give the prediction results of exploitation of the corresponding CVEs with the risk factors associated with it

Required Installs: Joern-0.3.1, Neo4j-community-2.1.8

Code Available at: https://github.com/Jokers-grin/VulScrape

Chapter 12: Conclusion

Working to develop a forecasting tool for cyber-attacks is a new experience to me. I have got the opportunity to learn various concepts about cyber security such as vulnerabilities, their exploitation, DeepLearning, neural network models (LSTM, GRU), LightGBM, syntax and semantic similarities. I hope this tool will help users be wary of their C/C++ source codes and predict exploit of the vulnerabilities in that part.

I have tried my level best to analyze requirements correctly and make a design to implement the tool as practical as possible. From this SRS report on the cyber-attack Forecast, the users will get a clear and easy view of the overall system. This SRS document can be used effectively to maintain the software development cycle. It will be very easy to conduct the whole project using SRS.

References

[1] Yong Fang, Yongcheng Liu, Cheng Huang, and Liang Liu. Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. PLOS One, 15(2):e0228439, 2020.

[2] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, S. Wang, and J. Wang. 2018. SySeVR: A framework for using deep learning to detect software vulnerabilities. CoRR abs/1807.06756.

[3] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "VulDeePecker: A deep learning-based system for multiclass vulnerability detection," IEEE Trans. Dependable Sec. Comput., vol. PP, pp. 1–1, 2019.

[4] LightGbm: https://lightgbm.readthedocs.io/en/latest/

[5] FastText: https://fasttext.cc/

[6] Roger S. Pressman. "Software Engineering A Practitioner's Approach." 7th Edition