



Masters Project Final Report March 2017

Project Title Developer Profiling for Secure Software Development

Student Name Lasith Tharindu Hettiarachchi

**Registration No. &
Index No.** 2014MIS007
14770072

Supervisor's Name Dr M.D.J.S. Goonathilaake

**Please Circle the
appropriate** **Master's Type
Program**

MIS Research Implementation

For Office Use Only

Developer Profiling for Secure Software Development

L. T. Hettiarachchi

2016



Developer Profiling for Secure Software Development

**A dissertation submitted for the Degree of Master of
Science in Information Security**

L.T. Hettiarachchi

**University of Colombo School of Computing
2016**



Declaration

“I, Lasith Tharindu Hettiarachchi hereby certify that this report does not incorporate, any material previously submitted for a degree or diploma in any University or higher educational institution in Sri Lanka or abroad without acknowledgement. Furthermore, it does not contain any material previously published or written by another person except where due reference is made in the text.”

Candidate: Mr. Lasith T. Hettiarachchi

.....

Signature

.....

Date

Supervisor: Dr M.D.J.S. Goonathilaake

.....

Signature

.....

Date

Acknowledgement

I would like to thank,

My parents, for all their support, and advice.

My supervisor, Dr. M.D.J.S. Goonathilaake for her continuous guidance, opinion and time.

All my lecturers at the University Of Colombo School Of Computing.

Abstract

This research is aimed to be a proof of concept for to identifying and predicting software developers who can be a security threat in a large multinational organization. The idea for this research was originated from the personal experience of the author.

Table of Contents

Declaration	iv
Acknowledgement	v
Abstract	vi
List of figures	xi
Acronyms and Abbreviations	xii
Chapter 1: Introduction	1
1.1 Preamble	1
1.2 Brief Overview of Project	3
1.2 Motivation	3
1.3 Scope of the Project	3
1.4 Goals of the Project	3
1.5 Limitation of the Project	4
1.6 Version History	5
1.7.1 Project Proposal Submission	5
1.7.2 Final Report	5
1.7 Overview of Report	6
Chapter 2: Literature Survey and Background	7
2.1 User Profiling for Computer Security [5]	7
2.3 User Modeling: Through Statistical Analysis and an Evolving Classifier.	9
Chapter 3: Requirements Specification	10
3.1 Functional Requirements	10
3.2 Non-functional Requirements	10
3.2.1 Product Requirements	10
Usability Requirements	10
Efficiency, Reliability and Robustness Requirements	10
Portability Requirements (Optional)	10
3.2.2 Organizational Requirements	11
Delivery Requirements	11
Implementation Requirements	11
3.2.3 External Requirements	11
Ethical and Legislative Requirements	11
Chapter 4: Design	12
4.1 Overall Architecture	12

4.1.1 Data extraction	12
4.1.2 Profile Generation	13
4.1.3 Identify vulnerable users	14
4.2 Design decisions	15
4.2.1 Approach	15
4.2.2 Identify sensible baseline	15
4.2.3 Source Code analyzers	16
4.2.4 Identifying suitable data source	16
Chapter 5: Implementation – Code Analyzer	19
5.1 Inject required parameters to the process	20
5.2 Prepare CSV document	20
5.3 Identify vulnerable developers.	21
5.4 Usage	22
5.4.1 Input	22
5.4.2 Output	24
Chapter 6: Implementation – Data Extractor	25
6.1 Extract data from the GitHub	25
6.2 Insert plug-in to a given the project	26
6.3 Executing the maven build	26
Chapter 7: Data Extraction	28
7.1 Hardware specifications	28
7.2 Repository search criteria	28
7.3 Execution and results	29
Chapter 8: Profile Generation	30
8.1 Feature used	30
8.2 Training Phase	30
8.2.1. Pre-processing stage	30
8.2.2. Training the neural network	30
8.2.3. Testing the trained system	30
8.3 Artificial Neural Networks (ANNs)	31
8.3.1. Reasons for adopting Artificial Neural Networks	33
8.3.2. MATLAB	33
8.3.2.1 Reasons for selecting MATLAB	34
8.3.3 Implementing the Neural Network	34
8.3.3.1 Input Layer	35

8.3.3.2 Hidden Layer	35
8.3.3.3 Output Layer	35
8.3.3.4 ANN Architecture	36
8.3.3.5 Categorization of Data Set	37
8.3.4. Network Training	37
8.2 Training Phase	40
Chapter 9: Testing, Evaluation and Validation	41
9.1 The Plan	41
9.1.1 Code Testing	41
9.1.1.1 White-box testing	41
9.1.1.2 Black-box testing	41
9.1.2 Manual validation	41
9.2 Summary of Code Testing	41
9.2.1 Summary of the test process	41
9.2.2 Summary of the test results	42
9.3 Summary of Manual Evaluation of Output	42
Chapter 10: Conclusions and Future Work	43
10.1 Problems Faced	43
10.1.1 Finding data source	43
10.1.2 Finding filtering tools	43
10.1.3 Time constrains	43
10.2 Lessons learned	43
10.2.1 Understanding code analyzers	43
10.2.2 Learning new technologies	43
10.2.3 Literature Reviewing	43
10.3 Deviations from original project plan	44
10.3.1 The name of the project	44
10.3.2 Scope of the project	44
10.4 Deficiencies in the final product	44
10.4.1 Platform Dependence	44
10.5 Extensions and Further Work	44
10.5.1 Complete original goals	44
10.5.2 Further validation	44
10.5.3 GUI style interfaced application	45
10.6 Final Conclusion	45

Appendix A	46
OWASP top 10 security threats	46
Appendix B	48
References	52

List of figures

Figure 1 : Data extraction process	12
Figure 2 : Profile generation process	13
Figure 3 : Identify vulnerable users	14
Figure 4 : GitHub overview	18
Figure 5 : Code analyzer process diagram	19
Figure 6 : Maven project output location	24
Figure 7 : GitHub search results	28
Figure 8 : Resource usage when data extractor running.....	29
Figure 9: Features prepared for the analysis.....	31
Figure 10: Biological Neuron Architecture [21]	31
Figure 11: Artificial Neuron Architecture	32
Figure 12: Typical ANN Architecture [22]	33
Figure 13: ANN Input Data Template	35
Figure 14: ANN Output Classes Template.....	35
Figure 15: Proposed ANN Architecture	37
Figure 16: ANN Training Performance.....	38
Figure 17 : Confusion Matrix for Validation	38
Figure 18: Training Confusion Matrix	39
Figure 19: Test Confusion Matrix	40

Acronyms and Abbreviations

ANN Artificial Neuron Network

RL Rule Learner

Chapter 1: Introduction

1.1 Preamble

With the growth of the software industry over past few decades, security has been one of the key areas that made into the spotlight [1]. Due to software design and operational issues, governments and industries are losing billions annually [2], [3]. As a result, there has been an emergence of a practice called “Secure Software Development”.

Private companies and government organizations involved in software development that cater to high stake industries like defense, health-care telecommunication, etc. are following additional security procedures throughout the software development life cycle. This will involve techniques like compartmentalization and categorization of all the information and projects based on the stakes involved and many other steps [12].

In this case, software developers and engineers have to play a major role because proportionate wise they represent the majority of the team and they are the ones who convert the business logic into a workable implementation. For an example, when a developer gets into a technical problem it is the normal practice to ask assistance from a fellow developer, if it has not been resolved, raise it on an online forum. But if the developer is in a so called secure software development project, he/she should be careful about the information that is being disclosed to the outside world or even to the fellow developer. It can be a simple question like selecting secure hashing algorithm, but the problem is how we can ensure that he/she is not disclosing any sensitive information to the third parties.

Implementing additional controls like limiting access to Internet or completely cutting down the access is not a sound solution because what if user discusses this sensitive information with outside people. We can't always rely on code reviews and code analysis of implementation which is more time consuming and costly most of the time.

The key to the solution for this kind of a problem is selecting a developer, who fulfils the following two major aspects,

1. Engineering aspects:

The ability to understand, design and address the operational flows

2. Behavioral aspects:

The ability to behave on a non-threatening way and withstand the pressure of doing so (based on the project security requirements there will be boundaries that need to be practiced. When this boundary becomes tighter, people will tend to succumb).

That is suitable for security level of a security related project.

1.2 Brief Overview of Project

With this study, author is trying to arrive at a technique that can be used to identify developers who can introduce a security vulnerability or security breach to the application which is being developed.

1.2 Motivation

Several motivational factors have inclined the author to attempt this project:

- Currently in most of the organizations developer behaviors are monitored and those data being captured/stored but cannot see any export system that takes use of this data tombs.
- The author has worked/been working in multinational organizations that provides information security services, protecting its customers' computers, networks and information assets from malicious activity such as cybercrime.
- Has been a witness to a data leakage that was caused by one of the developers.

1.3 Scope of the Project

- Although there are many forms of data sources that can be used for analysis in the project, we are only focusing on the coding patterns of the developers that are extracted from the source code repository.
- Nature of the languages that are used within the analysis will be limited due to time constrains; therefore, we are trying to focus on most popular programming languages within this scope.

1.4 Goals of the Project

The primary goal of this project (as stated in the project proposal) is to build a framework which can predict on developers' potentially harmful behaviors for the security integrity of the project.

1. Identify sensible baseline that dictates the criteria of identifying a developer who is meant to be security vulnerability.
2. Determining the appropriate data source that can be used within the project
3. Identify, if not, develop tools that can be used to filter out vulnerable developers from a selected data set
4. Develop a framework that can be used to process selected data set based on the tools

identify on the #3

5. With the use of existing data sources build up a profile that can be used to identify a potentially vulnerable developer.
6. Measuring the accuracy of these profiles.

1.5 Limitation of the Project

1. One of the main difficulties in a computer security related research is getting suitable data [4].Despite it's nearly impossible to convince a software company to grant access to their internal staff and projects to be used in our research. Therefore, we are proposing to use an online data source that can be utilized as an alternative.
2. Tools that are being used to analyses might have language dependencies.

1.6 Version History

1.7.1 Project Proposal Submission

Project proposal as submitted on the 20th April 2016 she suggested several revisions.

1.7.2 Final Report

This final document is the complete Final Report for the Individual Project (2014/MIS/007).

1.7 Overview of Report

Chapter 2: (Literature Survey and Background) reviews the background and existing literature related to the project. First, we will take a look at existing user profiling research papers. Then, we will review the existing techniques that we hope to use here

Chapter 3: (Requirements Specification) briefly presents the requirements of this project. Since this project is a research project, it does not contain a formal requirements specification that a classic software engineering project would contain. However, to avoid uncertainties, we present a set of functional and non-functional requirements.

Chapter 4: (Design) describes the design. It looks at architectural aspects of the initial design and describes the overall architecture as a whole, the design philosophy, the architecture and the basic design components. It also describes in detail how these design components are specifically designed.

Chapter 5: (Implementation – Code Analyzer) **Chapter 6:** (Implementation – Data extractor) Implementation of the code analyzers that **Find Bug** based maven plug-in and the Data extractor that use the Code analyzer we developed

Chapter 7: Data extraction. Use the Code Analyzer and the Data extractor developed and obtain a sample data set.

Chapter 8: (Testing, Evaluation and Validation) describes the testing and validation process. After an overview of the testing methods, it describes the validation of components developed

Chapter 9: (Conclusions and Future Work) begins with a summary of the results. It describes the problems faced during the project, the lessons learnt, deviations from the original project plan, deficiencies in the final product, extensions to the project and a critical appraisal of the system.

Chapter 2: Literature Survey and Background

2.1 User Profiling for Computer Security [5]

In this paper [5], a proof-of-concept has been validated based on the idea of profiling, queuing theory, and logistic regression modeling for intrusion and misuse detection in a definitive set of computer users, like, bank tellers, insurance and credit processing agents who would be expected to use their computers in a very similar and regular way. Their results were even though introductory, remained encouraging, suggesting that monitoring even very rough behavioral features might be effective in detecting intrusions and abuse in these particular classes of computer users.

Rather than digging deep into the operating system commands, users produce, they simply look at the busy period assembly of the sample path they spawn, i.e., when they are busy, how long they are busy, and approximately how active they are while busy. This leads to an agile and simple implementation, requiring only simple time-stamping and counting. Being mild and fast, another potential use, outside of user profiling, is for making a quick first pass over audit data.

Logistic regression is very similar to ANNs, who are sometimes used for IDS-MDS, but is statistically more rigorous and docile to conduct a comprehensive statistical analysis of model quality and sensitivity to various features.

The approach they have presented has certain boundaries.

1. Only discovers a partial portion of the space of likely behavior features. So, it is not designed to be used standalone to detect computer abuse, but as another “sensor” in an integrated, broad defense-in-depth security system. For example, the behavior of a bank teller can be correlated with customer arrivals to the bank’s lobby.
2. The approach does not deliver real-time detection, since it only makes its judgments when a session ends—although by simple variation one can picture a system that dynamically comes to a decision as features reveal themselves, e.g., while the length of the session is not known until it ends, the starting time, day, and interval since the last session are known immediately at the session start.
3. One of the main open questions to be addressed is to assess the effectiveness of the methodology in building profiles for groups of users, because the nature of their work has been predicted to have alike behaviors.

2.2 Combining Data Mining and Machine Learning for Effective User Profiling

[6]

The recognition of cellular cloning fraud is an interesting field to study. Fraud behavior changes frequently as outlaws adapt to detection techniques. To counter act this behavior, a fraud detection system should be adaptive as well. However, in order to build usage profiles, it must know which aspects of customers' behavior to profile. Traditionally, determining such aspects has involved a good deal of manual work, conjecturing useful features, building profilers and testing them. Determining how to combine them involves much trial-and-error as well.

In here, it combine data mining and constructive induction with more standard machine learning techniques to design methods for detecting fraudulent usage of cellular telephones based on profiling customer behavior and using a rule-learning program to uncover indicators of fraudulent behavior from a large database of cellular calls. These indicators are used to create profilers, which then serve as features to a system that combines evidence from multiple profilers to generate high-confidence alarms.

2.3 User Modeling: Through Statistical Analysis and an Evolving Classifier.

[7]

They (who are the researches) discussing approach for creating and automatic recognition of behavior profile of a user is combined with an evolving method to keep up to date profiles. The behavior of a computer is denoted in this research as the sequence of commands a specific user types during a period. This sequence is treated using statistical methods in order to create the matching user profile. Yet, a user profile is not usually fixed but rather it changes and develops. This paper defines briefly the model creation method and the evolving classifier, which are related with well-established off-line and on-line classifiers.

Also, this paper presents an approach for forming and identifying spontaneous behavior profiles of a computer user, and it is combined with an evolving method to keep up to date profiles. Since a user profile is usually not fixed but rather it changes, they have proposed a classifier which is able to keep up to date models based on Evolving Systems. This evolving classifier is one pass, non-iterative, recursive and it has the potential to be used in a collaborative mode; therefore, it is computationally very effective. Furthermore, an all important goal in this work is to give a general approach which can symbolize, manage and evolve contrary behaviors in a comprehensive environment. Hence, the proposed approach can be generalized to modeling, classifying and updating agent behaviors represented by a sequence of events.

The experimental results show that, using an appropriate subsequence length, EvCAB is very effective in both domains and it can perform almost as well as other well established offline classifiers in terms of correct classification on validation data. However, the proposed classifier is suitable in environments which it is necessary to cope with huge amounts of data and process streaming data quickly, because it does not need to store the entire data stream in the memory, and it is computationally simple and efficient as is

Chapter 3: Requirements Specification

3.1 Functional Requirements

As mentioned above, the main goal of this project is to build a framework that can predict on developers who can introduce a security threat to the company. Based on the goals of this project following requirements have been identified.

1. Identify sensible baseline criteria that can used to identify a developer who is vulnerable to security integrity and should be acceptable and reasonable.
2. Determining suitable data source that can be used in the project. Since it is impossible to acquire a private data source, we should find a data source that almost mirrors the qualities of source code repository of a software company.
3. Identify, if not develop tools that can be used to filter out security vulnerable developer from selected source code repository. In here we expect a tool or set of tools that can analyze source code or byte code and identify security vulnerabilities based on the baseline we provided.
4. Develop a framework that can be used to process selected data set. To arrive and make a reasonable judgment we must analyze reasonable amount of data therefore the developed framework should be memory efficient and process efficient.

3.2 Non-functional Requirements

The following non-functional requirements have been listed to counter certain issues that may arise.

3.2.1 Product Requirements

Usability Requirements

Human-Computer interfaces (for example, those used for moderation and listening) should conform to reasonable usability metrics. Since this mostly targeting technical persons command line interface would be sufficient.

Efficiency, Reliability and Robustness Requirements

Once the system is complete, it must satisfy efficiency, reliability and robustness constraints imposed by the domain.

Portability Requirements (Optional)

The system is planned to be developed on Ubuntu and expected to work smoothly in any

Linux based system mainly using Java.

3.2.2 Organizational Requirements

Delivery Requirements

Deadlines: This Individual Project (2014/MIS/007) needs to be finished before March 2017. The deadline applies to both the application and associated documentation.

Implementation Requirements

There are no direct requirements with regards to implementation software, hardware, etc. The selected implementation platform and components are acceptable to the generally acceptable Standards

The project must conform to generally accepted standards for coding, documentation, etc.

3.2.3 External Requirements

Ethical and Legislative Requirements

Composers have intellectual property rights to their compositions. If any outside material is used,

The legality of use must be verified. The legality of development and application software must also be verified.

Chapter 4: Design

4.1 Overall Architecture

4.1.1 Data extraction

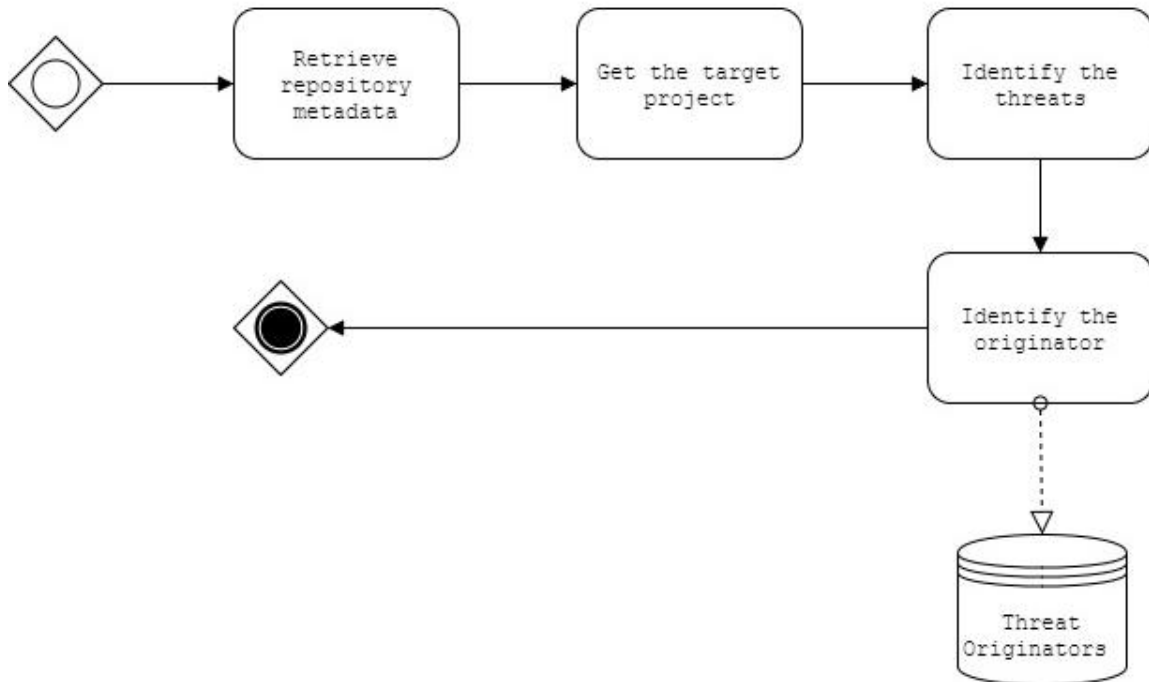


Figure 1 : Data extraction process

Since there is no already filtered out data set that is suitable for our research we need to filter out who are the security vulnerable developers from the selected data set.

Thus, for selected repository there can be a number of projects that can be categorized in to various categories based on different attributes like number of languages used, lines of code, number of committers etc.

And for the analysis of selected project there should be source code analyzers or static code analyzers that would be acting according to the specified baseline.

4.1.2 Profile Generation

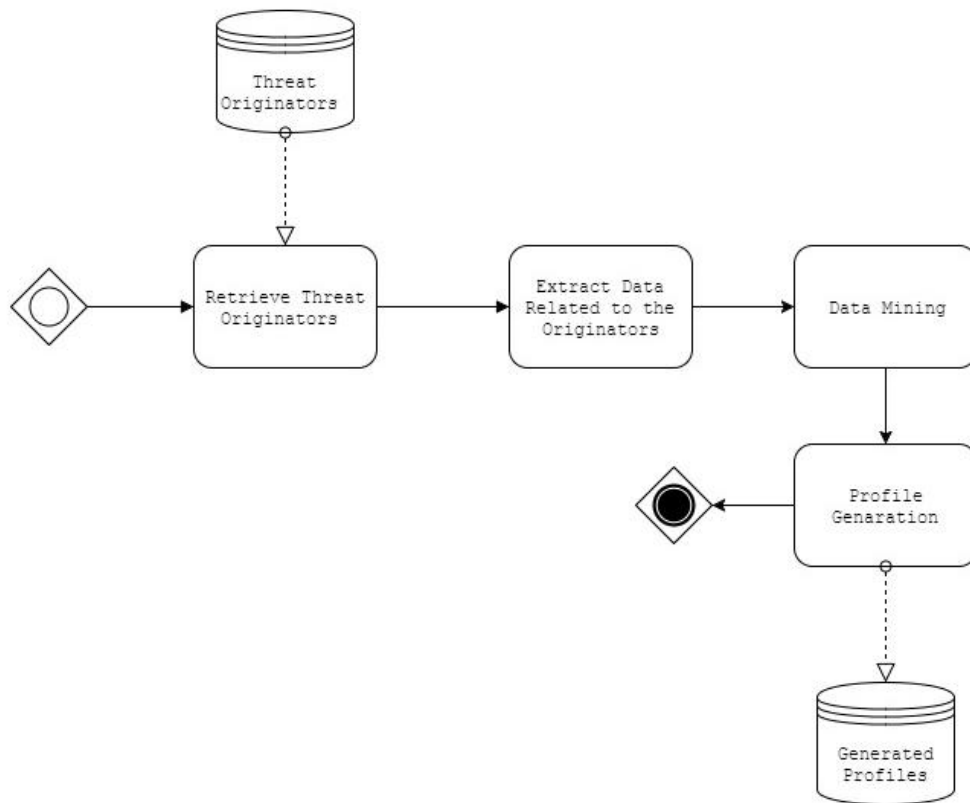


Figure 2 : Profile generation process

We expect to use the RL program (Clearwater & Provost 1990) [16], which is similar to other Meta-DENDRAL-style rule learners (Buchanan & Mitchell [17]; Segal & Etzioni [18]) searches for rules.

Since we are injecting only the users who originated vulnerability, we try to group them based on the vulnerability they have caused and certainty factor of a given rule.

4.1.3 Identify vulnerable users

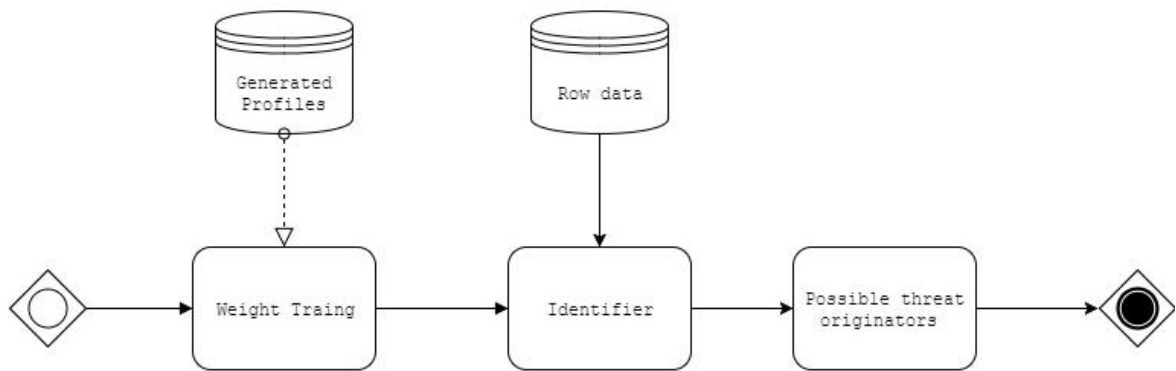


Figure 3 : Identify vulnerable users

With the use of the identified profiles in the previous phase, we will try to predict developers who can be vulnerable in future. Following criteria's have been identified for the evaluation.

- I. Determining the accuracy of profiles we have generated using training data set.
- II. The accuracy of predicting vulnerable developers using generated profile and effect of this early identification.

4.2 Design decisions

4.2.1 Approach

This research's main aim is to predict with reasonable probability, which can cause a security threat.

In this research we tried to identify a security vulnerable developers in simple terms we are trying to pick the ugly ones since to be in that category we only need to find one mistake.

4.2.2 Identify sensible baseline

One of the other aims in this study is to identify a developer who can be a security threat. Firstly, when identifying a security vulnerable developer we need to determine what the baseline criteria that need to be consider.

Since the baseline we do use should be unbiased and acceptable to the considerable length, thus we have looked in to the OWASP [12] with more than 42,000 volunteers [13] and considerable community recognition. Because of that reason, we have selected OWASP Top Ten Projects to determine our baseline where project goal is to raise awareness about application security by recognizing some of the most serious perils facing organizations.

4.2.3 Source Code analyzers

After the baseline was selected, we have looked on source code and byte code analyzers that can support the selected baseline criteria's. [15] Following tools and project have been identified as potential candidates during the initial analysis.

Tool	Language(s)	Available	Finds or Checks for
CheckmarxCxSAST	Java, JavaScript, PHP, C#, VB.NET, VB6, ASP.NET, C/C++, Apex, Ruby, Perl, Objective-C, Python, Groovy, HTML5, Swift, APEX, J2SE, J2EE	Checkmarx (Commercial)	All OWASP Top 10 and SANS 25 vulnerabilities and compliance with PCI-DSS, HIPAA, and MISRA requirements along with custom queries, all with a low rate of false-positives and easy to integrate throughout the SDLC.
SPARROW	C/C++, Java, JSP, JavaScript, C#, ASP(.NET), Objective-C, PHP, VB.NET, VBScript, HTML, SQL, XML	Fasoo (Commercial)	OWASP Top 10, SANS 25, CWE, CERT vulnerabilities, MISRA, efficient and effective issue management based on machine learning technology
Find Security Bugs	Java	LGPL (Free and Open Source)	OWASP TOP 10 and CWE coverage

Since first two tools are commercial tools, we have requested student license from them but the requests were rejected. Due to that reason we decided to go ahead with Find **Security Bugs** which is publicly available open source tool.

4.2.4 Identifying suitable data source

As mentioned above its nearly impossible to get hands on the source code repository from the private organizations therefore, we have looked for a public source code repository that can be accessed easily.

After engaging with some thorough analysis we have decided to go ahead with GitHub [14]. Apart from a rich data source, it provides an API that can be used retrieve repository statistics. It allows us to fetch data that GitHub uses for visualizing different types of repository activities.

On a beautiful April day on GitHub...

80k

Repositories
Updated

7k

People pushed
their first repository

4k

People forked
a repository
for the first time

3k

People created
their very first
pull request

30k

Issues created

12k

Pull Requests
created

And **6k** people signed up to join the fun.



Figure 4 : GitHub overview

Chapter 5: Implementation – Code Analyzer

After deciding go ahead with the **Find Security Bug** which is a plug-in developed for the **Find Bugs**. Which also have plug-in developed for different IDE's Eclipse, IntelliJ IDEA etc. and build tools like Maven, Gradle.

We have used JGit which is a Java implementation of Git version control system.

Based on the Maven **Find Bugs** plug-in we have implemented a plug-in which after executing this plug-in it will generate a CSV file with details that provide details of any developer who have committed a security vulnerable code retrieved from on any Git like source code system.

Following contains the important code snippets in the plug-in

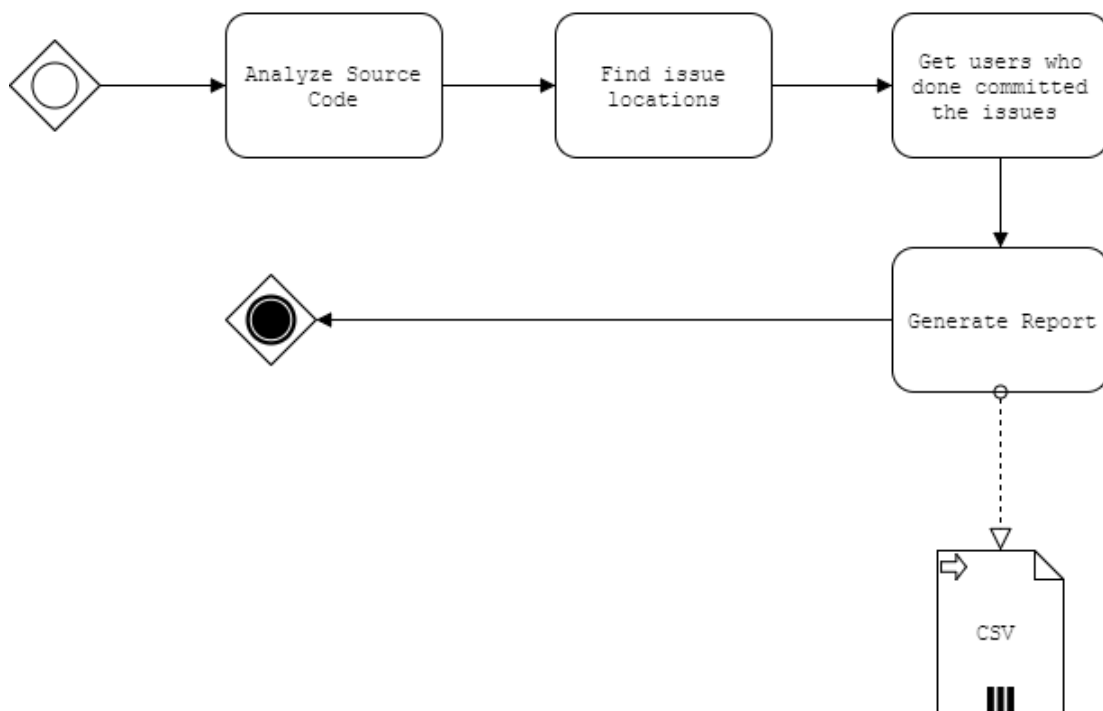


Figure 5 : Code analyzer process diagram

5.1 Inject required parameters to the process

```
/**
 * File of the Directory containing the source files.
 * injected by the maven start of the execution
 * @required
 */
@Parameter(defaultValue = '${project.build.sourceDirectory}', required = true)
File sourceFilesDirectory
```

```
/**
 * File of the Directory where the execution is started
 * @required
 */
@Parameter(defaultValue = '${session.executionRootDirectory}', required = true)
File executionRootDirectory
```

5.2 Prepare CSV document

```
File issuesBy = new File("${project.build.directory}/issuesBy.csv")
```

```
if (issuesBy.exists()) {
  issuesBy.delete()
}
issuesBy.getParentFile().mkdirs()
issuesBy.createNewFile()
xDocsReporter.generateReport(sourceFilesDirectory, executionRootDirectory, issuesBy)
```


5.3 Identify vulnerable developers.

```

public void generateReport(File sourceFilesDirectory, File executionRootDirectory, File
issuesBy) {
defstreamingMarkupBuilder = new StreamingMarkupBuilder()
streamingMarkupBuilder.encoding = "UTF-8"
defxdoc = {
mkp.xmlDeclaration()
log.debug("generateReportfindbugsResults is ${findbugsResults}")
BugCollection(version: getFindBugsVersion(), threshold:
FindBugsInfo.findbugsThresholds.get(threshold), effort:
FindBugsInfo.findbugsEfforts.get(effort)) {
findbugsResults.FindBugsSummary.PackageStats.ClassStats.each() { classStats ->
defclassStatsValue = classStats.'@class'.text()
defclassStatsBugCount = classStats.'@bugs'.text()
if (classStatsBugCount.toInteger() >0) {
bugClasses<<classStatsValue
    }
    }
    File gitFolder = new File("${executionRootDirectory.absolutePath}/.git")
    Repository repo = new FileRepositoryBuilder().setGitDir(gitFolder).build()
    BlameCommand blamer = new BlameCommand(repo)
    ObjectIdcommitID = repo.resolve("HEAD")
    bugClasses.each() { bugClass ->
log.debug("finish bugClass is ${bugClass}")
    file(classname: bugClass) {
findbugsResults.BugInstance.each() { bugInstance ->
if (bugInstance.Class.@classname.text() == bugClass) {
deftype = bugInstance.@type.text()
defcategory = bugInstance.@category.text()
defmessage = bugInstance.LongMessage.text()
defpriority = evaluateThresholdParameter(bugInstance.@priority.text())
defline = bugInstance.SourceLine.@start[0].text()
defbugPath = bugInstance.SourceLine.@sourcepath[0].text()
BugInstance(type: type, priority: priority, category: category, message: message,
lineNumber: ((line) ? line : "-1"))
if (category == "SECURITY") {
blamer.setStartCommit(commitID)
    String path =
"${executionRootDirectory.toURI().relativize(sourceFilesDirectory.toURI()).getPath()}${bu
gPath}"
    blamer.setFilePath(path)
    BlameResult blame = blamer.call()
    PersonIdentpersonIdent =
    blame.getSourceAuthor(Integer.parseInt(bugInstance.SourceLine.@start[0].text()))
    issuesBy.append(personIdent.toString() + "\r\n")
    }
    }
    }
    }
    }
    Error() {
findbugsResults.Error.analysisError.each() { analysisError ->

```

```

AnalysisError(analysisError.message.text())
    }
findbugsResults.Error.MissingClass.each() { missingClass ->
MissingClass(missingClass.text)
    }
    }
    Project() {
if (!compileSourceRoots.isEmpty()) {
compileSourceRoots.each() { srcDir ->
SrcDir(srcDir)
    }
    }
if (!testSourceRoots.isEmpty()) {
testSourceRoots.each() { srcDir ->
SrcDir(srcDir)
    }
    }
    }
    }
}
outputWriter<<streamingMarkupBuilder.bind(xdoc)
outputWriter.flush()
outputWriter.close()
}

```

5.4 Usage

After including our plug-in in the plug-in in the pom.xml file it will generate CSV file with the discovered details.

5.4.1 Input

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.sample.app</groupId>
<artifactId>findBug-sec-sample</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>findBug-maven-sample</name>
<url>http://maven.apache.org</url>
<reporting>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-project-info-reports-plugin</artifactId>
<version>2.9</version>
</plugin>
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>findbuggers-maven-plugin</artifactId>
<version>3.0.5-SNAPSHOT</version>

```

```
<configuration>
<effort>Max</effort>
<threshold>Low</threshold>
<xmlOutput>>true</xmlOutput>
<plugins>
<plugin>
<groupId>com.h3xstream.findsecbugs</groupId>
<artifactId>findsecbugs-plugin</artifactId>
<version>1.5.0</version><!-- Auto-update to the latest stable -->
</plugin>
</plugins>
</configuration>
</plugin>
</plugins>
</reporting>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

5.4.2 Output

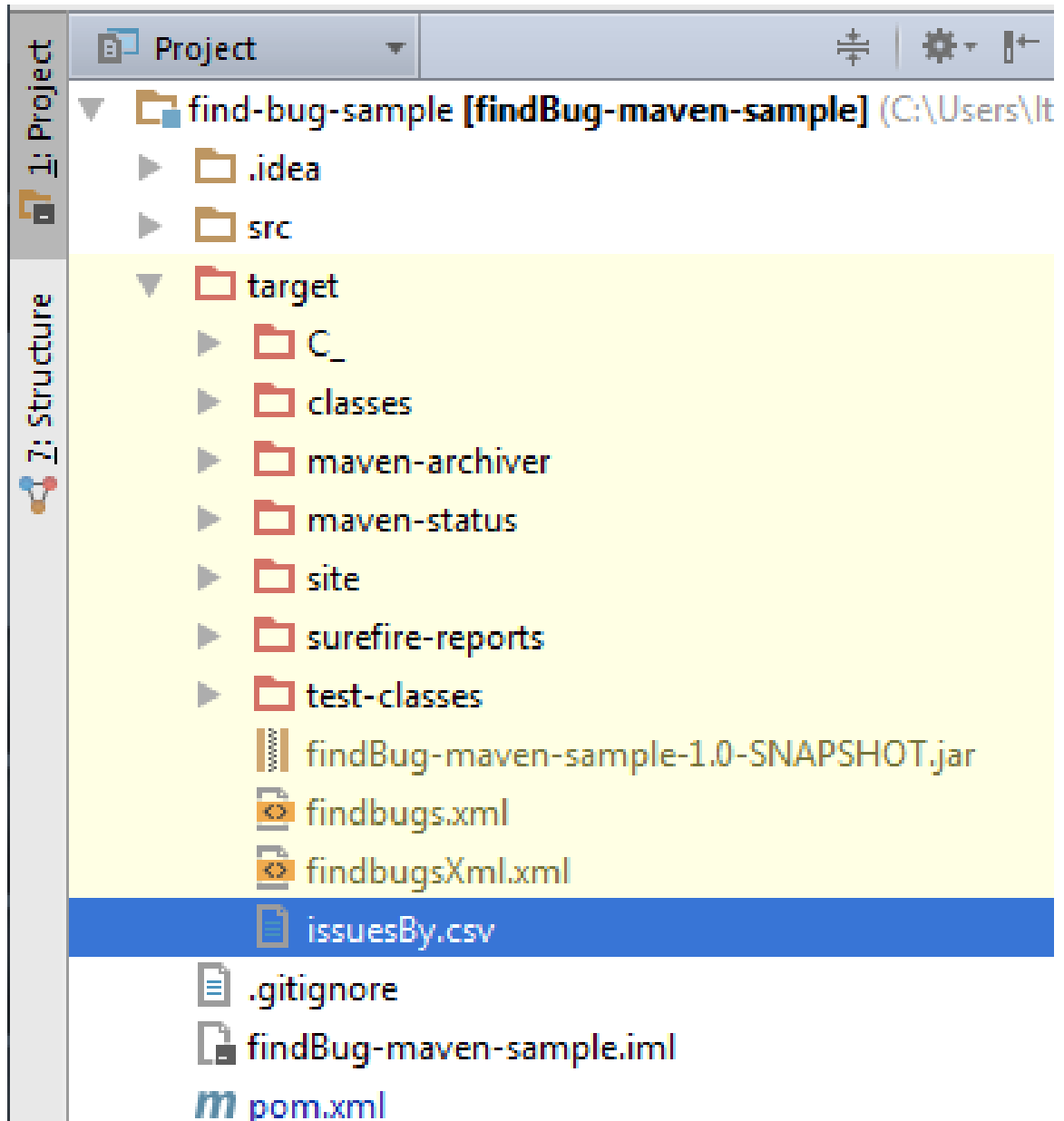


Figure 6 : Maven project output location

Chapter 6: Implementation – Data Extractor

Since in this project we intend to analyze large amount of row data to come up with a reasonable test data set following frame work have been developed to facilitate the process.

6.1 Extract data from the GitHub

This module will be responsible for extracting repository based on the given filter criteria's

```

/**
 * Initialize the client with the auth token
 * @paramoauthToken
 * @throws IOException
 */
GitClient(String oauthToken) throws IOException {
    this.oauthToken= oauthToken;
    this.github= getGithub();
}

private GitHub getGithub() throws IOException {
    GitHubBuildergitHubBuilder = new GitHubBuilder();
    gitHubBuilder.withOAuthToken(oauthToken);
    return gitHubBuilder.build();
}

/**
 * Get repository based on the filter criteria
 * @paramsize in Kb
 * @paramstars project ratings
 * @paramlanguage language that the project is developed
 * @return List of repository end points
 */
public List<String>getRepos(String size, String stars, String language) {
    return getRepos(github.searchRepositories()
        .size(size)
        .stars(stars)
        .language(language).sort(GHRepositorySearchBuilder.Sort.STARS).list());
}

/**
 * Get repository based on the filter criteria
 * @paramsearchBuilderPagedSearchIterable<GHRepository>
 * @return @return List of repository end points
 */
public List<String>getRepos(PagedSearchIterable<GHRepository>searchBuilder) {
    return searchBuilder.asList().stream().map(ghRepository -> {
        return ghRepository.gitHttpTransportUrl();
    }).collect(Collectors.toList());
}

```

6.2 Insert plug-in to a given the project

```
/**
 * Initiated pom modifier
 *
 * @param plugin Plugin to be inserted
 */
AddToPom(ReportPlugin plugin) {
this.plugin= plugin;
}
```

```
public void modifyPom(Model model, String baseDir) throws IOException {
    MavenXpp3Writer writer = new MavenXpp3Writer();
    writer.write(new FileOutputStream(new File(baseDir, "/pom.xml")), model);
}
```

6.3 Executing the maven build

```
/**
 * This method will execute the project in a given path and return a result
 *
 * @param projectPath File location where the project have been checked out
 * @return the status of the project
 */
public String buildThis(String projectPath) {
    String message = null;
    InvocationRequest request = getRequest(projectPath + "/pom.xml");
    InvocationResult result = null;
    try {
        result = runThis(request);
    } catch (MavenInvocationException e) {
        e.printStackTrace();
    }

    if (result.getExitCode() != 0) {
    if (result.getExecutionException() != null) {
        message = "Failed to build project : " + result.getExecutionException();
        //throw new PublishException( );
    } else {
        message = "Failed to build project : " + result.getExitCode();
        //throw new PublishException("Failed to publish site. Exit code: ");}
    }
    }
    return message;
}

/**
 * Invoke the maven build
 *
 * @param request
 * @return
 */
private InvocationResult runThis(InvocationRequest request) throws
MavenInvocationException {
```

```
    Invoker invoker = new DefaultInvoker();
    //invoker.setMavenHome(new File("C:\\apache-maven-3.3.3"));
    invoker.setMavenHome(new File(mavenHome));
    InvocationResult result = null;
    return invoker.execute(request);
}

/**
 * Prepare command to be executed
 *
 * @param projectPath location where the project is checked out
 * @return InvocationRequest
 */
private InvocationRequest getRequest(String projectPath) {
    InvocationRequest request = new DefaultInvocationRequest();
    request.setPomFile(new File(projectPath + "/pom.xml"));
    request.setGoals(Arrays.asList("clean", "compile", "site"));
    return request;
}
```

Chapter 7: Data Extraction

For the data extraction using tools we developed in the Chapter 5 and 6 we have used Dell inspiron 5559 laptop with following specification and search criteria's

7.1 Hardware specifications

<i>Processor</i>	5th Generation Intel® Core™ i7-5500U Processor (4M Cache, 2.50 GHz)
<i>Operating Systems</i>	Ubuntu 16.04 LTS
<i>Memory</i>	8 GB DDR3L 1600 MHz
<i>Hard Drive</i>	1000GB SATA

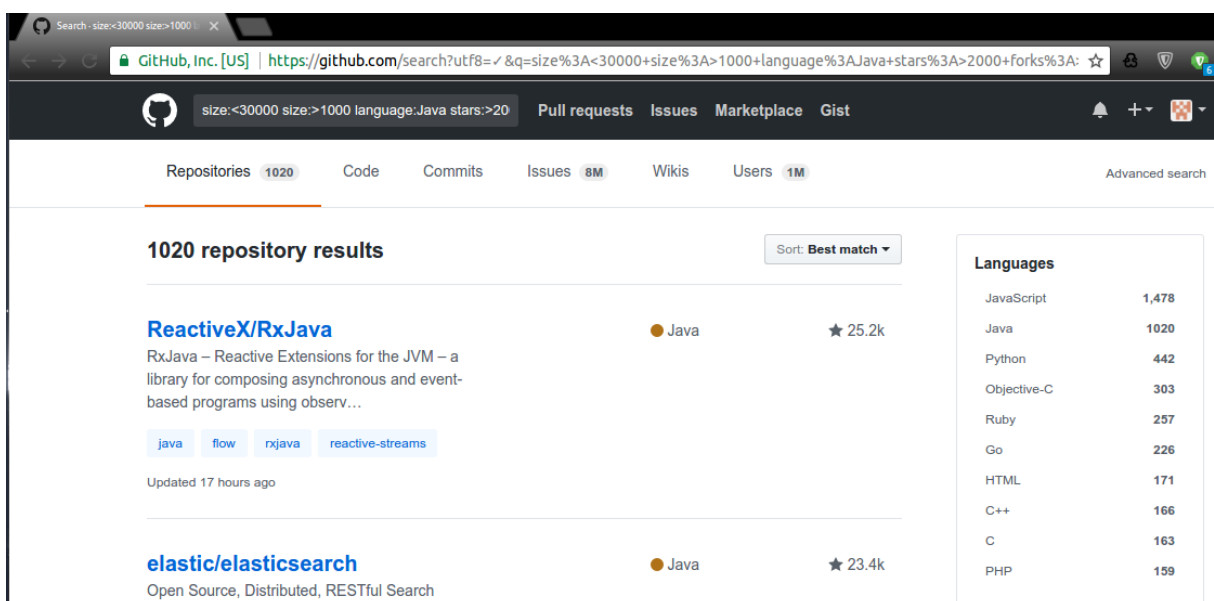
And the laptop was connected to the internet connection which had effective bandwidth of 10mb/s.

7.2 Repository search criteria

Since we have limited resources to use we targeted for relatively small repositories that have high user involvement. We have used following search criteria's for filtering out repositories.

<i>Maximum repository size</i>	Less than 30mb
<i>Programming language</i>	Java
<i>Stars (user rating)</i>	More than 2000

Based on these search criteria's we have got 1020 repositories for the analysis.



Search - size:<30000 size:>1000 | X

GitHub, Inc. [US] | <https://github.com/search?utf8=✓&q=size%3A<30000+size%3A>1000+language%3AJava+stars%3A>2000+forks%3A:>

size:<30000 size:>1000 language:Java stars:>20 Pull requests Issues Marketplace Gist

Repositories 1020 Code Commits Issues 8M Wikis Users 1M Advanced search

1020 repository results Sort: Best match

ReactiveX/RxJava ● Java ★ 25.2k
RxJava – Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observ...

java flow rxjava reactive-streams

Updated 17 hours ago

elastic/elasticsearch ● Java ★ 23.4k
Open Source, Distributed, RESTful Search

Languages

JavaScript	1,478
Java	1020
Python	442
Objective-C	303
Ruby	257
Go	226
HTML	171
C++	166
C	163
PHP	159

Figure 7 : GitHub search results

7.3 Execution and results

To analyze 1020 repositories it took around 5 hours and provides a data set of 1747 mistake from 100 plus developers.

One of the main problems we face is the high resource usage of the extractor when analyzing large set of data.



Figure 8 : Resource usage when data extractor running

Chapter 8: Profile Generation

8.1 Feature used

We have gone for following features for a given user when building up a profile.

- Country
- Number of Public REPOS
- Number of Public GIST
- Number of Followers of the user
- Number of users user following
- Issues that have been reported for the user.

8.2 Training Phase

Data extracted from the GitHub have been used in the Training phase. An artificial neural network (ANN) is used to profile each developer and their corresponding security vulnerability. There are three main sub phases under this phase.

1. Pre-processing stage
2. Training the neural network
3. Testing the trained system

8.2.1. Pre-processing stage

The gathered data will be processed in to a format that is in line with the neural network input format. Pre selected features were taken into the input template.

8.2.2. Training the neural network

Processed data sets were mapped into the corresponding target classes (vulnerable or not) and trained till the system receives a minimum Mean Squared Error (MSE) and a good correlation value between system outputs and targets.

8.2.3. Testing the trained system

Separate data set was used to evaluate the system for its accuracy on classifying the features for their corresponding vulnerability.

	A	B	C	D	E	F	G	H
1	akhmerov	Berlin	camunda	25	1	16	10	18
2	null	Russia	[x]	89	13	14	27	42
3	allwefanta	China	null	42	33	468	9	43
4	null	Austria	null	30	1	10	6	14

Figure 9: Features prepared for the analysis

8.3 Artificial Neural Networks (ANNs)

ANNs is one type of networks that see the nodes as “Artificial Neurons.” An artificial neuron is a computational model inspired by the natural neurons. Natural neurons accept signals through synapses located on the dendrites or membrane of the neuron. When the received signals’ strength is strong enough (surpass a certain threshold), the neuron will be activated and emits a response signal through the axon (figure 10). This cycle will be continued in order to activate neurons for signal transmission [20].

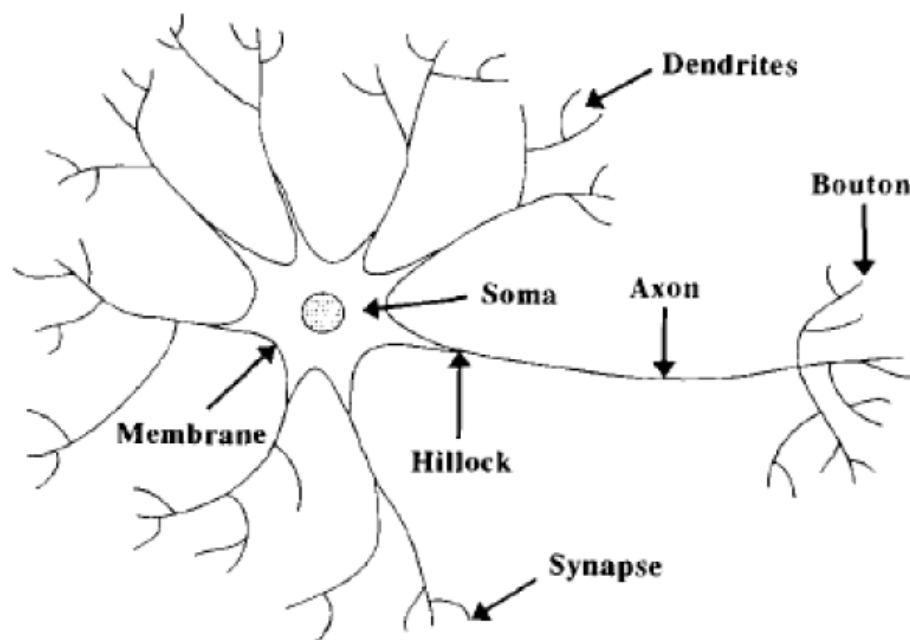


Figure 10: Biological Neuron Architecture [21]

The complexity of real neurons is highly abstracted when constructing artificial neurons. These basically consists of inputs (like synapses), which are multiplied by weights (strengths of the respective signals), and then computed by a mathematical function which determine the activation of the neuron (figure 11). Another function computes the output of artificial neuron. This output sometimes depends on a defined threshold value. Artificial neural networks combine artificial neurons in order to process information.

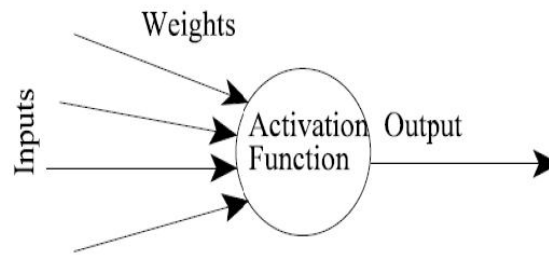


Figure 11: Artificial Neuron Architecture

When the weight of an artificial neuron is high, the stronger the input which is multiplied by it weights can also be negative, thus it is said that the signal is inhibited by the negative weight. The computation complexity of the neuron will be depended on the weights. By adjusting the weights of an artificial neuron, desired output can be achieved. When there are hundreds or thousands of neurons, it would be quite complicated to find by hand all the necessary weights. There are algorithms which can adjust the weight of the ANN in order to obtain the desired output from the network.

The number of types of ANN and their uses are very high. Since the first neural model by McCulloch and Pitts in 1943, different types of ANN models have been implemented. The differences of them may be the functions, accepted values, topology, learning algorithm, etc. The ANN architecture is clearly illustrated in figure 12.

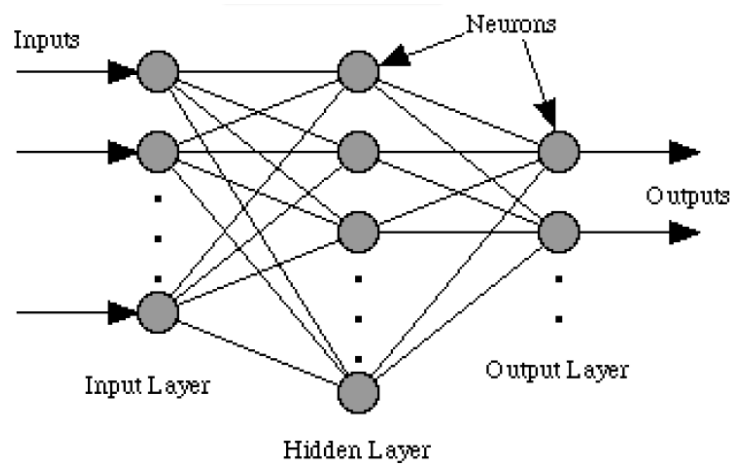


Figure 12: Typical ANN Architecture [22]

8.3.1. Reasons for adopting Artificial Neural Networks

ANN approach suits for the training of the system due to the following reasons [23].

1. Instances are represented by many attribute-value pairs.

The target function to be trained is defined by the instances that can be described by a vector of predefined features. This study holds the input nodes as a vector comprised with selected features. Furthermore, these input attributes can be highly correlated or independent from one another and at the same time these input values can be any real values.

2. The target function output may be discrete-valued, real-valued, or a vector of real or discrete valued attributes.

Targets of the trained model where the user vulnerable or not where each level is comprised with 2 digits. The function output was a discrete-valued or real-valued vector.

3. The training attributes may contain errors.

Even though training attributes contain errors, learning methods are quite robust to the noise in training data set.

4. Fast evaluation of the learned target function is generally required.

Although ANN learning times are relatively long (depends on the data set), evaluating a subsequent instance is typically very fast.

8.3.2. MATLAB

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. This tool can be used to analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable

users to explore multiple approaches and reach a solution faster. MATLAB can be used for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing [24].

8.3.2.1 Reasons for selecting MATLAB

As stated above, MATLAB is an interactive system for conducting numerical computations. MATLAB is a well-established mathematical package which uses highly respected computational models which the user can be confident of. As per our research requirement, MATLAB Neural Network Tool box provides functions and apps for modeling complex nonlinear systems that are not easily modeled with a closed-form equation [24].

8.3.3 Implementing the Neural Network

Initial creation of ANN consists of few mandatory steps such as,

- a) Define input layer.
- b) Process hidden layer according to number of input nodes.
- c) Define output layer in correspondence with input layer.

MATLAB Neural Network Pattern Recognition (NNPR) tool is used for building up the ANN. NNPR tool uses two-layered feed-forward architecture in constructing the ANN. The network will be trained with scaled conjugate gradient back propagation algorithm [25]. According to [86], “Multilayer perceptions can form arbitrarily complex decision boundaries and represent any Boolean function. The development of the back-propagation learning algorithm for determining weights in a multilayer perception has made these networks the most popular among researchers and users of neural networks.” Furthermore, back-propagation algorithm is a gradient-descent method to minimize the mean squared error (average squared difference between outputs and targets). The function, “mapminmax” scales inputs and targets and they fall in the range between -1 to 1 . MATLAB script for ANN implementation is included in the Appendix B.

8.3.3.1 Input Layer

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	3	14	6	1	17	10	14	13	13	6	14	6	11	19
2	25	89	42	30	11	39	8	12	12	19	6	43	65	0
3	1	13	33	1	7	0	3	0	0	0	0	22	15	0
4	16	14	468	10	6	31	68	1	1	0	3	1053	381	0
5	10	27	9	6	0	0	49	2	2	1	4	56	2	0
6	18	42	43	14	44	28	39	39	28	23	44	23	16	8
7														

Figure 13: ANN Input Data Template

From the data set we have discovered 105 records have been selected as the training data set and the 21 records have been selected as the evaluation dataset. Figure 13 illustrates the input data template in the input layer.

8.3.3.2 Hidden Layer

Deciding the number of neurons in the hidden layer is an important stage in constructing the overall network architecture. Even though these layers do not directly interact with external environment, they highly influence the final output [26].

Using only a few neurons in the hidden layer will result in under fitting. This occurs when there are few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Adapting too many nodes in the hidden layer can result in over fitting. Over fitting occurs when the neural network has so much information processing capacity, while training set is not enough to train all the neurons in the hidden layers.

There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers such as,

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be $2/3$ the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

With referring to the above facts, we have chosen 5 hidden layer nodes for training. Based on the results, number of hidden nodes could be increased or decreased in order to achieve an optimized output. In our scenario, 5 hidden nodes provided the optimum training accuracy.

8.3.3.3 Output Layer

Two output nodes were included in the architecture. Two digits were used to formulate the corresponding output class as shown in figure 14.

CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV
0	0	0	0	1	0	0	1	1	1	0	0	0	1
1	1	1	1	0	1	1	0	0	0	1	1	1	0

Figure 14: ANN Output Classes Template

8.3.3.4 ANN Architecture

The proposed network architecture will be illustrated under figure 15.

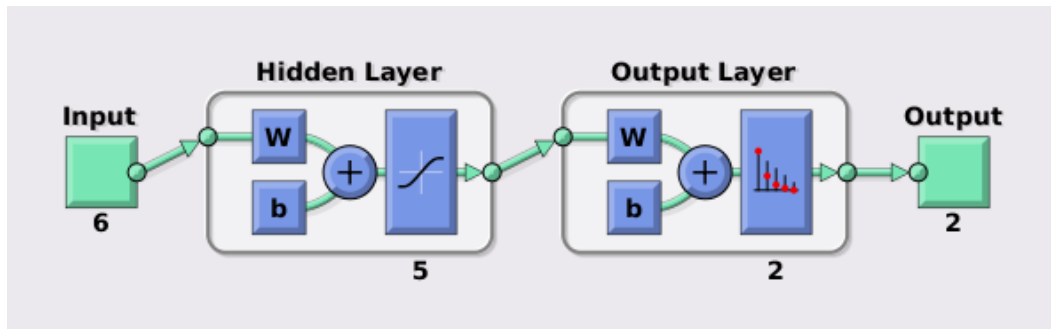


Figure 15: Proposed ANN Architecture

8.3.3.5 Categorization of Data Set

As mentioned, altogether there are 126 test samples. From those, 21 sample sets were allocated to be used at evaluation phase. Therefore, 105 sample sets were used for system training.

That data set is fed in to the neural network with the following proportions.

- Training: 95 samples (90%)
- Validation: 5 samples (5%)
- Testing: 5 samples (5%)

Testing and validation required comprised with 5 samples. We have focused more on the training data set since the system evaluation will be done through a different data set apart from the training data set.

8.3.4. Network Training

Network performance is an important factor regarding the efficiency and effectiveness of the network. The network performance will be discussed as follows.

a) Neural Network Training Performance

The training stopped when the validation set's mean square error came across to its lowest value. According to figure 16, network has a 0.063792 best validation performance at epoch 32. Validation data set was used to fine tune the network architecture. All the validation data sets have been correctly classified with 0% of error rate (figure 17).

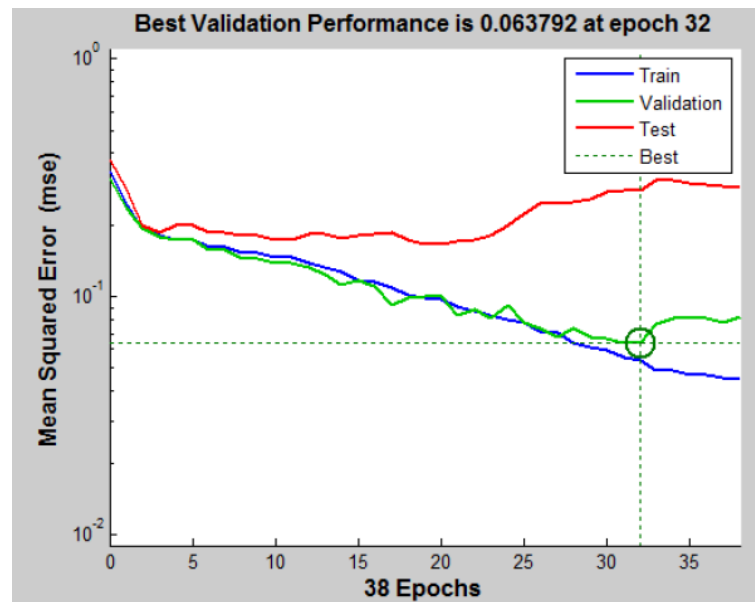


Figure 16: ANN Training Performance

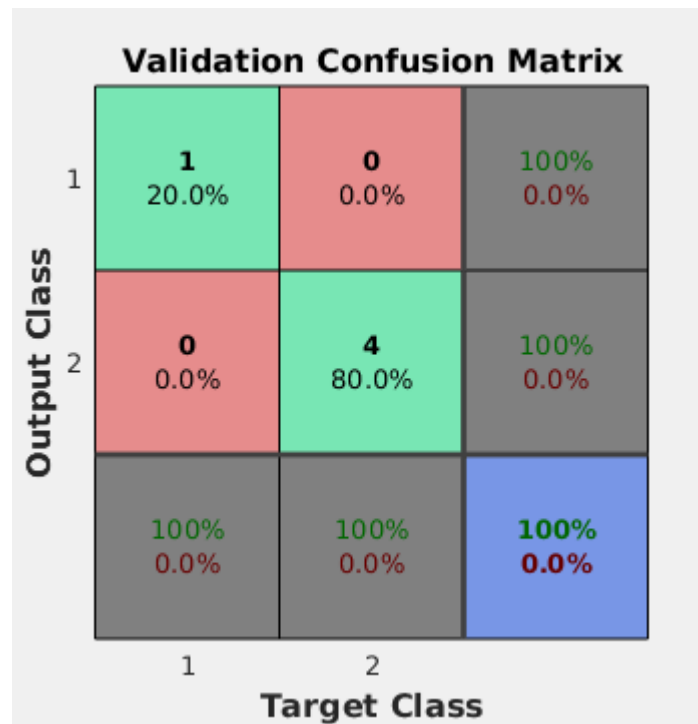


Figure 17 : Confusion Matrix for Validation

b) Network Training Confusion Matrix

Overall system training information is included in figure 18. According to the matrix, 87% (91 samples) of the total training data set (105 samples) is trained accurately. This is the optimum classification occurred where number of hidden neurons exist at 5.

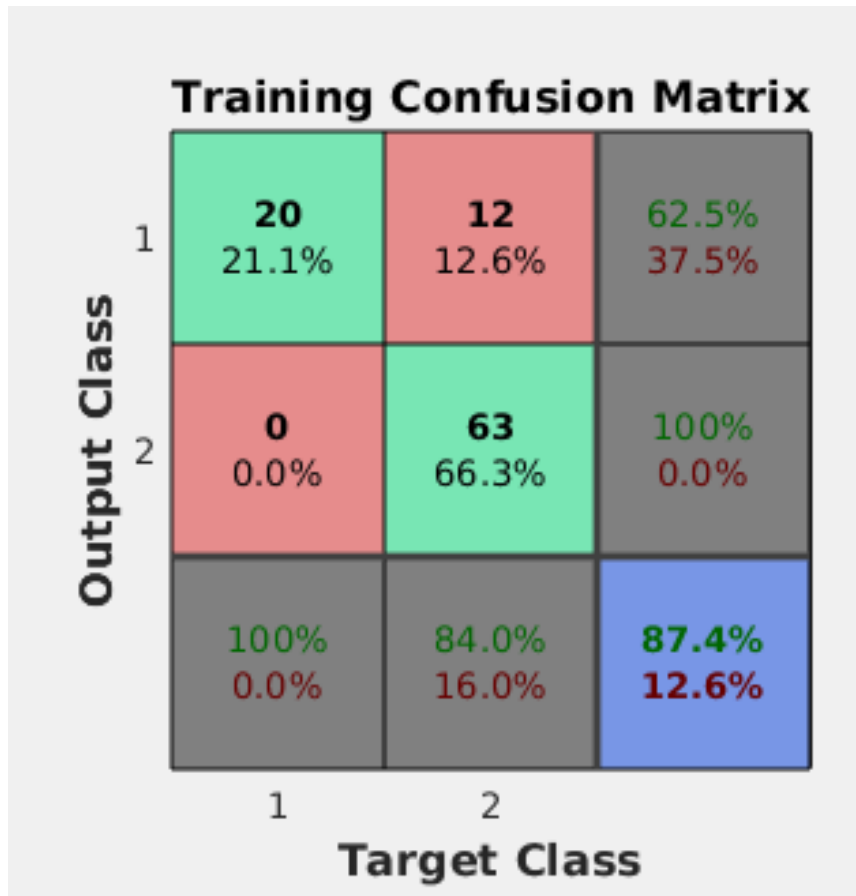


Figure 18: Training Confusion Matrix

8.2 Training Phase

Trained ANN was tested with 21 separate data samples. These 21 unseen data samples were classified into corresponding output classes with an accuracy level of 69.6%. Figure 19 shows the confusion matrix of the test data samples. Hence, from the evaluation we can conclude that the ANN is able to classify the user's into that he/she is vulnerable or not with an accuracy rate of 69.6%.

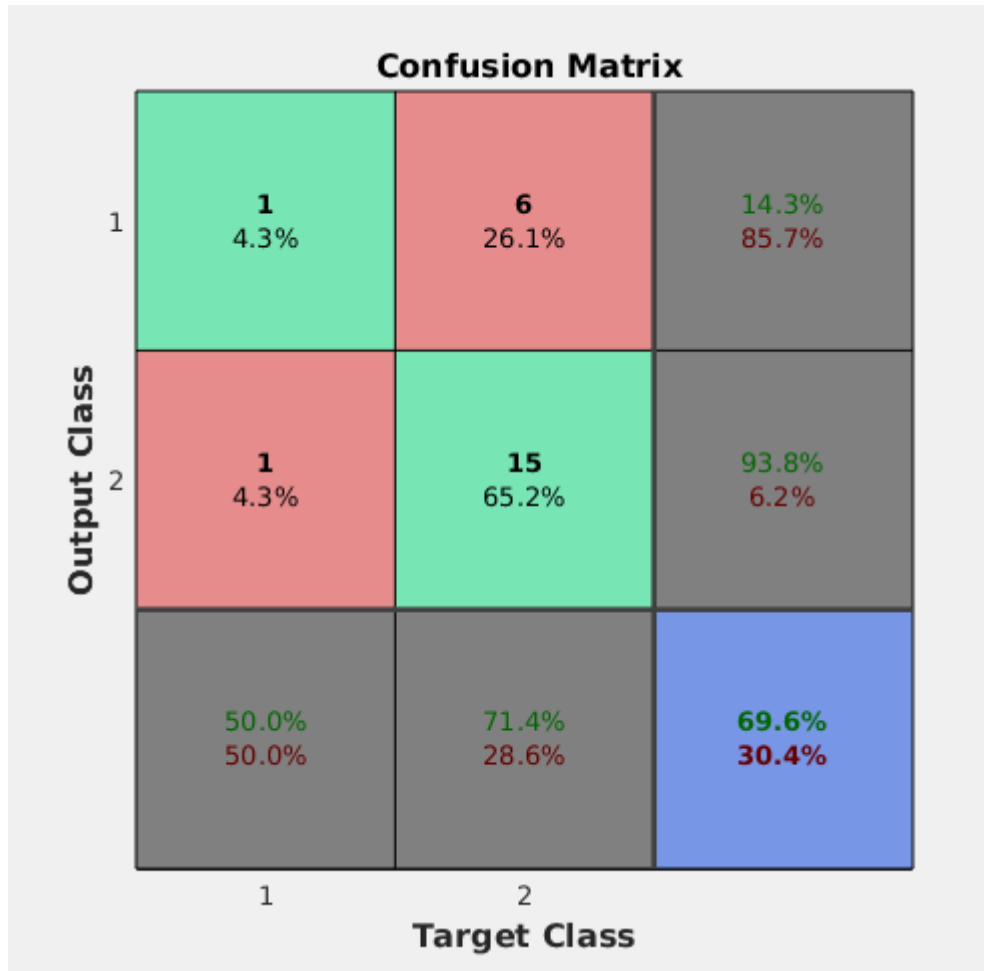


Figure 19: Test Confusion Matrix

Chapter 9: Testing, Evaluation and Validation

Due to the time constraints we were able to fully deliver only up to the #4 goal because of that the test plan only focus on the #3 and #4 goals were are the technical components

9.1 The Plan

Testing, Evaluation and Validation of the system was carried out as following

- First, the implemented deliverables were tested (Code Testing).
- The systems output will be validated manually for the conformance of the accuracy.

9.1.1 Code Testing

Since this project is not a conventional software development project, a conventional testing procedure need not be followed. Most of the implemented code follows directly from the design.

Hence, the likelihood of semantic and logical errors is low. Almost all syntax can be directly detected by the development IDE (IntelliJ IDEA: Community edition 2016.3.5). Still, we will be performing the below described tests to verify that the components can be integrated, functionality is met and non-functional requirements are satisfied.

9.1.1.1 White-box testing

- Test control structures and paths for correct implementation
- Test the interfaces of functions
- Test the dependencies between packages

9.1.1.2 Black-box testing

- Test each generator for functionality (to make sure it produces the required output)
- Test the performance of the procedures (mainly the speed)

9.1.2 Manual validation

To validate the validity of the implementations output sample will be inspected.

9.2 Summary of Code Testing

9.2.1 Summary of the test process

Before get on the actual code testing, the full source code was carefully inspected for errors. A few errors were discovered. The review also provided the chance to confirm coding conventions and comments etc. The white box testing process was based on the design of the code implementation. Path testing, control structure testing was done while running the

software in debug mode. To enable testing definite sets of input data, simple test drivers were executed. Dependencies between packages were tested in a similar manner. Black-box testing was done incrementally in a bottom-up manner, maintained by drivers.

9.2.2 Summary of the test results

As expected the white-box testing uncovered no serious errors or bugs. Control structures and pathways were free of error. Interfaces between functions were error-free. There were no errors emerging from dependencies between packages. The makers performed according to specification. Similarly, the input options resulted in the expected output results. The speed and efficiency of the system was good.

9.3 Summary of Manual Evaluation of Output

After inspecting the sample of the out of the system we discovered that there are false positive and false negative scenarios but not in staid percentage.

Chapter 10: Conclusions and Future Work

10.1 Problems Faced

10.1.1 Finding data source

Finding a proper data source was the first obstacle we faced in this research due to the sensitive nature of the data we interested in this research we were unable to find access to a private company's source code repository because of this reason we have to turn in to a public data source.

10.1.2 Finding filtering tools

Our first two selections for the data filtering were the most suitable but due to licensing issues we have to drop them and develop our own tool for the filtering of the data using open source software's.

10.1.3 Time constrains

Because of the above two problems we faced from the intended questions we aimed to address only fist four of them were able to address at the end.

10.2 Lessons learned

10.2.1 Understanding code analyzers

Although the author had previously studied both source and byte code analysers and also he was practically used such applications this was the first he have got know their internal architectures and technical make up.

10.2.2 Learning new technologies

The author had the opportunity to work with several new computer technologies and related concepts. For example, he gained much knowledge about the source code systems and code analysis tools

10.2.3 Literature Reviewing

The actual design and execution was preceded by a very inclusive literature review that consisted of reviewing a large body of information. This process enabled the author to acquire skill set reading through material quickly, classifying data and appropriately documenting them.

10.3 Deviations from original project plan

10.3.1 The name of the project

The name of the project was changed from “User Behavioral analysis for Computer Security” to “Developer Profiling for Secure Software Development”. The new name is more representative of the scope of the project and the name as a whole more commercially relevant. This is important for an application with potential commercial value.

10.3.2 Scope of the project

Original project was intended to capture various data sources to extract data like network logs, bug tracking systems, etc. but due to data limitation and the time constrains the scope was cut down to the current extent.

10.4 Deficiencies in the final product

Although the final product meets the specified requirements satisfactorily, it might be said to be slightly deficient in a performance and usability

10.4.1 Platform Dependence

This project was written in java which has cross platform capability but since author has used the Ubuntu for the implementation there are dependencies that tied them Linux based systems.

10.5 Extensions and Further Work

10.5.1 Complete original goals

Due the problems faced the following goals were unable to achieved

1. With the use of existing data sources build up a profile that can be used to identify a potentially vulnerable developer.
2. Measuring the accuracy of these profiles.

10.5.2 Further validation

Since current study has done on a public data source. If the implementation can be tested on actual intended target user group and it will help to identify deficiencies.

10.5.3 GUI style interfaced application

It can be made more accessible for users by building an application with a graphical user interface. The complex nature of a potential user interface was one reason why it was not attempted as part of this project.

10.6 Final Conclusion

The author is confident that the research has achieved reasonable success and produces a strong foundation to further continuation. It has several interest features and implements many novel concepts. It can also be considered as an excellent foundation for developing advance featured source code mining applications in the future. The author hopes that this implementation will be used for professionals in computer security and data analytic fields as well.

Appendix A

OWASP top 10 security threats

[8]

1. Injection

Injection flaws, such as SQL, OS, and LDAP injection occur when un-trusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

2. Broken authentication and session management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

3. Cross-site scripting

XSS flaws occur whenever an application takes un-trusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

4. Insecure direct object reference

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

5. Security mis-configuration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

6. Sensitive data exposure

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra

protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

7. Missing function level access control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

8. Cross-site request forgery

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

9. Using components with known vulnerabilities: Heart bleed and Shellshock in action

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

10. Un-validated redirects and forwards

Web applications frequently redirect and forward users to other pages and websites, and use un-trusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Appendix B

```
% Solve a Pattern Recognition Problem with a Neural Network

% Script generated by Neural Pattern Recognition app

% Created 10-Jul-2017 09:41:34

%

% This script assumes these variables are defined:

%

% train_in - input data.

% train_out - target data.

x = train_in;

t = train_out;

% Choose a Training Function

% For a list of all training functions type: help nntrain

% 'trainlm' is usually fastest.

% 'trainbr' takes longer but may be better for challenging problems.

% 'trainscg' uses less memory. Suitable in low memory situations.

trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network

hiddenLayerSize = 5;

net = patternnet(hiddenLayerSize, trainFcn);

% Choose Input and Output Pre/Post-Processing Functions

% For a list of all processing functions type: help nnprocess

net.input.processFcns = {'removeconstantrows','mapminmax'};

net.output.processFcns = {'removeconstantrows','mapminmax'};
```

```
% Setup Division of Data for Training, Validation, Testing

% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.divideParam.trainRatio = 90/100;
net.divideParam.valRatio = 5/100;
net.divideParam.testRatio = 5/100;

% Choose a Performance Function

% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy'; % Cross-Entropy

% Choose Plot Functions

% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network

[net,tr] = train(net,x,t);

% Test the Network

y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
```

```
% Recalculate Training, Validation and Test Performance

trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
%figure, plotroc(t,y)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
```

```
genFunction(net,'myNeuralNetworkFunction');  
  
y = myNeuralNetworkFunction(x);  
  
end  
  
if (false)  
  
    % Generate a matrix-only MATLAB function for neural network code  
    % generation with MATLAB Coder tools.  
  
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');  
  
    y = myNeuralNetworkFunction(x);  
  
end  
  
if (false)  
  
    % Generate a Simulink diagram for simulation or deployment with.  
    % Simulink Coder tools.  
  
    gensim(net);  
  
end
```

References

- [1] Cybersecurity Ventures. 2016. The Cybersecurity Market Report covers the business of cybersecurity, including market sizing and industry forecasts, spending, notable M&A and IPO activity, and more. Cybersecurity Ventures. [ONLINE] Available at: <http://cybersecurityventures.com/cybersecurity-market-report/>. [Accessed 19 April 2016].
- [2] Design, programming, by iMarc. More info at <http://imarc.net>. 2016. The Cybersecurity Act of 2015 Is a Necessary Stake in the Ground | RSA Conference. [ONLINE] Available at: <http://www.rsaconference.com/blogs/the-cybersecurity-act-of-2015-is-a-necessary-stake-in-the-ground>. [Accessed 19 April 2016].
- [3] THE INTERNATIONAL INFORMATION SYSTEMS SECURITY CERTIFICATION CONSORTIUM The 2015 (ISC)2 Global Information Security Workforce Study In-text: (The International Information Systems Security Certification Consortium) Your Bibliography: The International Information Systems Security Certification Consortium,. The 2015 (ISC)2 Global Information Security Workforce Study. 2016. Web. 19 Apr. 2016.
- [4] R. Clandos, "Eye on Cybercrime," IEEE Security & Privacy Magazine, Vol. 1, No. 4, July/August 2003.
- [5] D.L. Pepyne, J. Hu, and W. Gong, "User Profiling for Computer Security," Proc. Am. Control Conf., pp. 982-987, 2004.
- [6] Fawcett, T., & Provost, F. (1996). Combining Data Mining and Machine Learning for Effective User Profiling. Retrieved September 25, 2016, from aaii, <https://www.aaii.org/Papers/KDD/1996/KDD96-002.pdf>
- [7] Iglesias J.A., Ledezma A., and Sanchis A. (2007), 'Sequence Classification Using Statistical Pattern Recognition', Proc. Int'l Conf. Intelligent Data Analysis (IDA), pp. 207-218
- [8] van der Stock, A., Gonçalves, I.R. and Correa, J. (2015) OWASP top Ten cheat sheet. Available at: https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet (Accessed: 27 September 2016).

- [9] Exchange, S. (2016) Stack exchange API. Available at:
<https://api.stackexchange.com/docs> (Accessed: 27 September 2016).
- [10] Home - find security bugs (no date) Available at: <http://find-sec-bugs.github.io>
(Accessed: 27 September 2016).
- [11] J. Whitehead, “An Introduction to Logistic Regression,”
<http://personal.ecu.edu/whiteheadj/data/logit/>.
- [12] <http://ijikm.org/Volume5/IJIKMv5p083-099Talib453.pdf>
- [13] OWASP (no date) Available at: <https://www.owasp.org> (Accessed: 2 March 2017).
- [13] OWASP (2017) in Wikipedia. Available at: <https://en.wikipedia.org/wiki/OWASP>
(Accessed: 3 March 2017).
- [14] Comparison of source code hosting facilities (2017) in Wikipedia. Available at:
https://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities (Accessed: 5
March 2017)
- [15] Source code security Analyzers (2008) Available at:
https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html (Accessed: 5 March
2017).
- [16] Foster John Provost, Daniel N. Hennessy. 2017. Distributed Machine Learning: Scaling
up with Coarse-grained Parallelism. [ONLINE] Available at:
<https://pdfs.semanticscholar.org/0154/40723d8cecab626507138bb6fb7c948ded69.pdf>.
[Accessed 15 February 2017].
- [17] Quinlan, J. Mach Learn (1986) 1: 81. doi:10.1023/A:1022643204877
- [18] Richard Segal , Oren Etzioni. 2017. Learning Decision Lists Using Homogeneous Rules.
[ONLINE] Available at: <https://www.aaai.org/Papers/AAAI/1994/AAAI94-094.pdf>.
[Accessed 15 February 2017].
- [19] D. Abrams, Introduction to Regression, [Online], 2007, Available:
http://dss.princeton.edu/online_help/analysis/regression_intro.htm#slr
- [20] C. Gershenson, Artificial Neural Networks for Beginners, [online], n.d.,
<http://arxiv.org/ftp/cs/papers/0308/0308031.pdf>

[21]E. Y. Li, Artificial neural networks and their business applications, in Information & Management, vol. 27, no. 5, 1994, pp. 303-313.

[22]W. Christian, L. Robert, R. Brown, J. Darby, Modelling Ranunculus Presence in the Rivers Test and Itchen Using Artificial Neural Networks, [online], 2000, Available: <http://www.geocomputation.org/2000/GC016/Gc016.htm>.

[23] N. Joakim, Machine Learning-Major Approaches, [online], n.d., Uppsala University and Växjö University, Sweden, Available:<http://stp.lingfil.uu.se/~nivre/gslt/approaches07ho.pdf>.

[24] The MathWorks Inc., *MATLAB*, [online], 2014, http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf

[25] M. F. Møller, A scaled conjugate gradient algorithm for fast supervised learning. Neural networks, vol. 06, no. 04, 1993, pp. 525-533.

[26]J. Heaton, The Number of Hidden Layers, [online], 2014, <http://www.heatonresearch.com/node/707>.