XBS WEB VULNERABILITY SCANNER

A project submitted in partial fulfillment of the
requirements for the award of degree of

Bachelor of Technology
in
Computer Science and Engineering

By

Chenna Bhargava (N100950)

Pangi Susanth (N091130)

Shaik Jafar Sadhik (N100775)

Under the Supervision of

Mr. Sunil Singh

Department of Computer Science and Engineering
Rajiv Gandhi University of Knowledge Technologies – Nuzvid
Nuzvid, Krishna – 521202

April, 2016

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)
RGUKT-NUZVID, Krishna Dist - 521202
Tele Fax : 08656 – 235557/235150

CERTIFICATE OF COMPLETION

This is to certify that the work entitled, "**XBS WEB VULNERABILITY SCANNER"** is the bonafied work of ***Chenna Bhargava (N100950), Pangi Susanth (N0901130), Shaik Jafar Sadhik (N100775)*** carried out under our guidance and supervision for the partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in the department of Computer Science and Engineering under RGUKT IIIT Nuzvid. This work is done during the academic session August 2015 – May 2016, under our guidance.

Mr. Sunil Singh
Project Supervisor
Lecturer Dept. of CSE
RGUKT IIIT Nuzvid, Nuzvid

Mr. K K Singh
Head of Department
Assistant Professor Dept. of CSE
RGUKT IIIT Nuzvid, Nuzvid

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)
RGUKT-NUZVID, Krishna Dist - 521202
Tele Fax : 08656 – 235557/235150

CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, "**XBS Web Vulnerability Scanner"** is the bonafide work of ***Chenna Bhargava (N100950), Pangi Susanth (N091130), Shaik jafar Sadhik (N100775),*** and here by accord our approval of it as a study carried out and presented in a manner required for its acceptance in the partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn, as a recorded in this project. It only signifies the acceptance of this project for the purpose for which it has been submitted**.**

Mr. Sunil Singh

Project Supervisor
Lecturer Dept. of CSE
RGUKT IIIT Nuzvid, Nuzvid

Examiner

Project Examiner
Lecturer Dept. of CSE
RGUKT IIIT Nuzvid, Nuzvid

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)
RGUKT-NUZVID, Krishna Dist - 521202
Tele Fax : 08656 – 235557/235150

DECLARATION

We, ***Chenna Bhargava, ID No: N100950, Pangi Susanth, ID No: N091130, Shaik Jafar Sadhik, ID No: N100775*** hereby declare that the project report entitle "**XBS Web Vulnerability Scanner"** done by us under the guidance of **Mr. Sunil Singh Sir** is submitted for the partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** the academic session August 2015 – April 2016 at RGUKT – Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references.

The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Date : _____ **Chenna Bhargava [N100950]**

Place :_____ **Pangi Susanth [N091130]**

Shaik Jafar Sadhik [N100775]

ACKNOWLEDGEMENT

We would like to express our profound gratitude and deep regards to my guide ***Mr. Sunil Singh sir*** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project.

We are extremely grateful for the confidence bestowed in us and entrusting our project entitled "XBS Web Vulnerability Scanner".

At this juncture we feel deeply honored in expressing our sincere thanks to him for making the resources available at right time and providing valuable insights leading to the successful completion of our project.

We would like to thank RGUKT Nuzvid Director, faculty and staff for their valuable suggestions and discussions.

Last but not least we thank almighty, and we place a deep sense of gratitude to our family members and our friends who have been constant source of information during the preparation of this project work.

Chenna Bhargava(N100950),
Pangi Susanth(N091130),
Shaik Jafar Sadhik(N100775),

ABSTRACT

An approach to the security testing of web applications consists of using black-box web vulnerability scanner. Black-box web vulnerability scanner is a tool that can be used to identify security issues in web applications. This tool is often marketed as "point-and-click pentesting" tool that automatically evaluate the security of web applications with little or no human support. This tool access a web application in the same way users do, and, therefore, have the advantage of being independent of the particular technology used to implement the web application. However, this tool needs to be able to access and test the application's various components

This is the tool that can crawl a web application to enumerate all the reachable pages and the associated input vectors (e.g., HTML form fields and HTTP GET parameters), generate specially-crafted input values that are submitted to the application, and observe the application's behavior (e.g., its HTTP responses) to determine if a vulnerability has been triggered. If triggered, it will analyze the problem or vulnerability and show the patch or solution for it.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

XSS	Cross Site Scripting
SQLI	SQL Injection
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
OWASP	Open Web Application Security Project
IDE	Integrated Development Environment

CHAPTER 1

INTRODUCTION

1.1. Problem Definition

Web application vulnerabilities, such as cross-site scripting and SQL injection, are most pressing security problems on the Internet today. In fact, web application vulnerabilities are widespread, accounting for the majority of the vulnerabilities reported in the Common Vulnerabilities and Exposures database, they are frequent targets of automated attacks and, if exploited successfully, they enable serious attacks, such as data breaches and drive-by-download attacks. In this scenario, security testing of web applications is clearly essential.

1.2. Web Vulnerabilities

Here, we will briefly describe some of the most common vulnerabilities in web applications (for further details, the interested reader can refer to the OWASP Top 10 List, which tracks the most critical vulnerabilities in web applications.

1.2.1. SQL Injection:

A SQLI vulnerability results from the application's use of user input in constructing database statements. The attacker invokes the application, passing as an input a (partial) SQL statement, which the application executes. This permits the attacker to get unauthorized access to, or to damage, the data stored in a database. To prevent this attack, applications need to sanitize input values that are used in constructing SQL statements, or else reject potentially dangerous inputs. This is the second most serious vulnerability on the OWASP Top 10 List.

1.2.2. Cross-Site Scripting (XSS):

XSS vulnerabilities allow an attacker to execute malicious JavaScript code as if the application sent that code to the user. This is the first most serious vulnerability of the OWASP Top 10 List.

1.2.3. HTTP Parameter Pollution Attacks:

HTTP Parameter Pollution attacks (HPP) have only recently been presented and discussed and have not received much attention so far. An HPP vulnerability allows an attacker to inject a parameter inside the URLs generated by a web application. The consequences of the attack depend on the application's logic, and may vary from a simple annoyance to a complete corruption of the application's behavior.

1.3. Importance of web vulnerability scanner

Securing a company's web applications is today's most overlooked aspect of securing the enterprise. Hacking is on the rise with as many as 75% of cyber attacks done through the web and via web applications. Most corporations have secured their data at the network level, but have overlooked the crucial step of checking whether their web applications are vulnerable to attack.

1.4. Black-Box Testing

Black-box testing is a method of software testing that examines the functionality of application without peering into its internal structures or workings. This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

Specific knowledge of the applications code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

In penetration testing, black-box testing refers to a methodology where an ethical hacker has no knowledge of the system being attacked. The goal of a black-box penetration test is to simulate an external hacking or cyber warfare attack. The vulnerability scanner that we are developing will have no idea of development code of the web application it is scanning. It will crawl the website search for the links and forms and test the web applications with the sample attacks.

1.5. Web Crawling

Primary task of web vulnerability scanning is web crawling. Web crawler plays key role in vulnerability scanning. It will find the all possible links and input fields to make sample attack.

1.5.1. Fundamentals of a Web Crawler

Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawlers work:

1. Download the Web page.

2. Parse through the downloaded page and retrieve all the links.

3. For each link retrieved, repeat the process.

1.5.2. Crawling Challenges

Crawling is arguably the most important part of a web application vulnerability scanner, if the scanner's attack engine is poor, it might miss a vulnerability, but if it's crawling engine is poor and cannot reach the vulnerability, then it will surely miss the vulnerability.

HTML Parsing: Malformed HTML makes it difficult for web application scanners to crawl web sites.

Infinite Web Site: It is often the case that some dynamically-generated content will create a very large (possibly infinite) crawling space or some web applications may have links to the other websites, those websites may have links of some other websites. This will results an infinite loop of crawling.

CHAPTER 2

TYPES OF VULNERABILITY SCANNERS

Vulnerability scanners can be divided broadly into two groups: network-based scanners that run over the network, and host-based scanners that run on the target host itself.

2.1. Network based Scanners

A network-based scanner is usually installed on a single machine that scans a number of other hosts on the network. It helps detect critical vulnerabilities such as misconfigured firewalls, vulnerable web servers, risks associated with vendor-supplied software, and risks associated with network and systems administration.

Different types of network-based scanners include:

1. Port Scanners that determine the list of open network ports in remote systems;

2. Web Server Scanners that assess the possible vulnerabilities (e.g. potentially dangerous files or CGIs) in remote web servers;

3. Web Application Scanners that assess the security aspects of web applications (such as cross site scripting and SQL injection) running on web servers. It should be noted that web application scanners cannot provide comprehensive security checks on every aspect of a target web application. Additional manual checking (such as whether a login account is locked after a number of invalid login attempts) might be needed in order to supplement the testing of web applications.

2.2. Host-Based Scanners

A host-based scanner is installed in the host to be scanned, and has direct access to low- level data, such as specific services and configuration details of the host's operating system. It can therefore provide insight into risky user activities such as using easily guessed passwords or even no password. It can also detect signs that an attacker has already compromised a system, including looking for suspicious file names, unexpected new system files or device files, and unexpected privileged programs. Host-based scanners can also perform baseline (or file system) checks. Network-based scanners cannot perform this level of security check because they do not have direct access to the file system on the target host.

CHAPTER 3

LITERATURE REVIEW

- Trustwave's Spiderlabs estimates that "82% of web applications are vulnerable to XSS" (2013).
- WhiteHat Security report says that "XSS regained the number one spot for being the most common vulnerability" (2014).
- Cenzic confirms that "XSS leads the [most popular vulnerabilities] list in terms of frequency of occurrence" (2014).
- CWE by MITRE warned about the problem in 2011 saying that "XSS is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications" (2014).
- Over 90% of XSS vulnerabilities can be exploited in such a manner that even advanced users and IT people will not suspect anything.
- The structure and architecture of over 70% of web applications allows creation of a sophisticated XSS exploit that will perform several fully-automated consecutive actions, giving full administrative access to the attacker at the end.
- More than 95% of XSS vulnerabilities can be used to perform sophisticated drive- by-download attacks infecting users who just open a harmless-looking URL they trust.
- SSL certificate and HTTPS connection to the website have absolutely no impact on web application security and can never prevent XSS attack.

The following image is the list of vulnerabilities and their percentage of attacks with respect to different programming languages

Vulnerability class by language (percentage)

	ASP	ColdFusion	.NET	Java	Perl	PHP
Cross-Site Scripting	49	46	35	57	67	56
Information Leakage	29	24	44	15	11	17
Content Spoofing	5	4	5	8	6	7
SQL Injection	8	11	6	1	3	6
Cross-Site Request Forgery	2	2	2	4	4	2
Insufficient Transport Layer Protection	0.8	1	0.9	1	0.3	4
Abuse of Functionality	0.3	6	0.3	0.9	0.5	0.2
HTTP Response Splitting	0.9	3	0.8	2	0.8	0.3
Predictable Resource Location	0.1	0.1	0.0	0.2	0.1	1
Brute Force	0.7	0.3	1	2	0.8	1
URL Redirector Abuse	0.7	0.4	0.5	1	1	0.9
Insufficient Authorization	0.2	0.3	0.5	0.9	1	0.2
Fingerprinting	0.3	0.1	0.5	0.6	0.3	0.1
Session Fixation	0.2	0.3	0.2	0.6	0.1	0.3
Directory Indexing	-	-	0.0	0.0	-	0.3

Ref :
http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf

CHAPTER 4

WEB VULNERABILITY SCANNING PROCESS

4.1. Scanner modules

4.1.1. Crawling Module

The crawling component is seeded with a set of URLs, retrieves the corresponding pages, and follows links and redirects to identify all the reachable pages in the application and save them in the database. In addition, the crawler identifies all the input points to the application, such as the parameters of GET requests, the input fields of HTML forms and give them as input to the attacking module.

4.1.2. Attacking Module

The attacker module analyzes the URLs discovered by the crawler and the corresponding input points. Then, for each input and for each vulnerability type for which the web application vulnerability scanner tests, the attacker module generates values that are likely to trigger a vulnerability. For example, the attacker module would attempt to inject JavaScript code when testing for XSS vulnerabilities, or strings that have a special meaning in the SQL language, such as ticks and SQL operators, when testing for SQL injection vulnerabilities. Input values are usually generated using heuristics or using predefined values, such as those contained in one of the many available XSS and SQL injection cheat-sheets.

4.1.3. Analysis Module

The analysis module analyzes the pages returned by the web application in response to the attacks launched by the attacker module to detect possible vulnerabilities and to provide feedback to the other modules. For example, if the page returned in response to input testing for SQL injection contains a database error message, the analysis module may infer the existence of a SQL injection vulnerability.

4.1.4. Solution or Patch Module

The solution or patch module will provide the patches or solutions for the vulnerabilities found in the analysis module.

CHAPTER 5

OPERATING ENVIRONMENT

5.1. Scanner Developed System Configuration

OPERATING SYSTEM - WINDOWS 8
PROCESSOR - Intel core PENTIUM
RAM SIZE - 4 GB
HARD DISK DRIVE - 100GB

5.2. Minimum Required System Configuration

OPERATING SYSTEM - WINDOWS 8
PROCESSOR - Intel core PENTIUM
RAM SIZE - 2 GB
HARD DISK DRIVE - 20 GB

CHAPTER 6

TECHNOLOGIES

Web vulnerability Scanner needs browser simulator to work like browser for crawling and providing input to input fields. To create browser environment some java and python technologies are used. Following technologies are used in the development of the web vulnerability scanner.

6.1. Jsoup Library

Jsoup is a java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data,, using the best of DOM, CSS, and jquery-like methods. Jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do. Jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree.

6.2. JDBC Driver

A JDBC Driver is a software component enabling a java application to interact with a database. To connect with individual databases JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

6.3. Jython Library

Jython is an implementation of the Python programming language designed to run on the Java platform. It is the successor of JPython. Jython programs can import and use any Java class. Except for some standard modules, Jython programs use java classes instead of python modules. Jython includes almost all of the modules in the standard python programming language distribution, lacking only some of the modules implemented originally in C. For example, a user interface in Jython could be written with Swing, AWT or SWT.

6.4. ClientForm Module

ClientForm is a python module for handling HTML forms on the client side, useful for parsing HTML forms, filling them in and returning the completed forms to the server. It developed from a port of Gisle Aas per module HTML::Form, form the libwww-perl library, but the interface is not the same.

6.5. Setuptools Package

Setuptools is a fully featured, actively maintained, and stable library designed to facilitate packaging python projects where packaging includes: python package and module definitions, distribution package metadata, test hooks, project installation, platform specific details, python 3 support.

6.6. Mechanize Module

Mechanize implements the urllib2.OpenerDirector interface. Browser objects have state, including navigation history, HTML form state, cookies, etc. The set of features and URL schemes handled by Browser objects is configurable. The library also provides an API that is mostly compatible with urllib2. Features include: ftp, http and file, URL schemes, browser history, hyperlink and HTML form support, HTTP cookies, HTTP EQUIV and Refresh, Referer header, robots.txt redirections, proxies, and basic and digest HTTP authentication.

6.7. BeautifulSoup Library

BeautifulSoup is a python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

6.8. NetBeans IDE

NetBeans is a software development platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers. The NetBeans IDE is primarily intended for development in JAVA, but also supports other languages, in particular PHP, C/C++ and HTML5.

6.9. SQL Server 2003

SQL Server allows multiple clients to use the same database concurrently. SQL Server provides two modes of concurrency control: pessimistic and optimistic concurrency. When pessimistic concurrency control is being used, SQL Server controls concurrent access by using locks.

CHAPTER 7

FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

7.1. Functional Requirements

- Scanner will use settings of default browser of the current operating system. So, it is required to user must keep the browser setting as default.
- Proxy authentication is not developed with scanner. So the testing url must be accessible without the proxy authentication.
- User can provide any valid url of the website for vulnerability scanning.
- User can find solutions to the vulnerabilities found.

7.2. Non-Functional Requirements

7.2.1. Analysis Module

- The scanners will request just the url of the website from the user that is user want to test. Remaining work will be done by the scanner.

7.2.2. Reliability

- MySQL provides a good deal of data reliability in terms of data storage and retrieval.
- JSoup provide excellent and reliable crawling functionalities.
- Python technologies can handle cookies, sessions while accessing web page.
- Java is efficient in crawling and Python is efficient in browser simulation.

7.2.3. Supportability:

- The Scanner is designed as it can work with any default browser of operating system.
- The Scanner is supported on environment which can operate python and java operations.

7.2.4. Security:

- The scanner is developed with most secured language java.

7.2.5. **Maintainability**:

- The maintenance of the scanner is required when the new attack samples needs to be integrated with the scanner. It is very easy to add attack samples to the code.

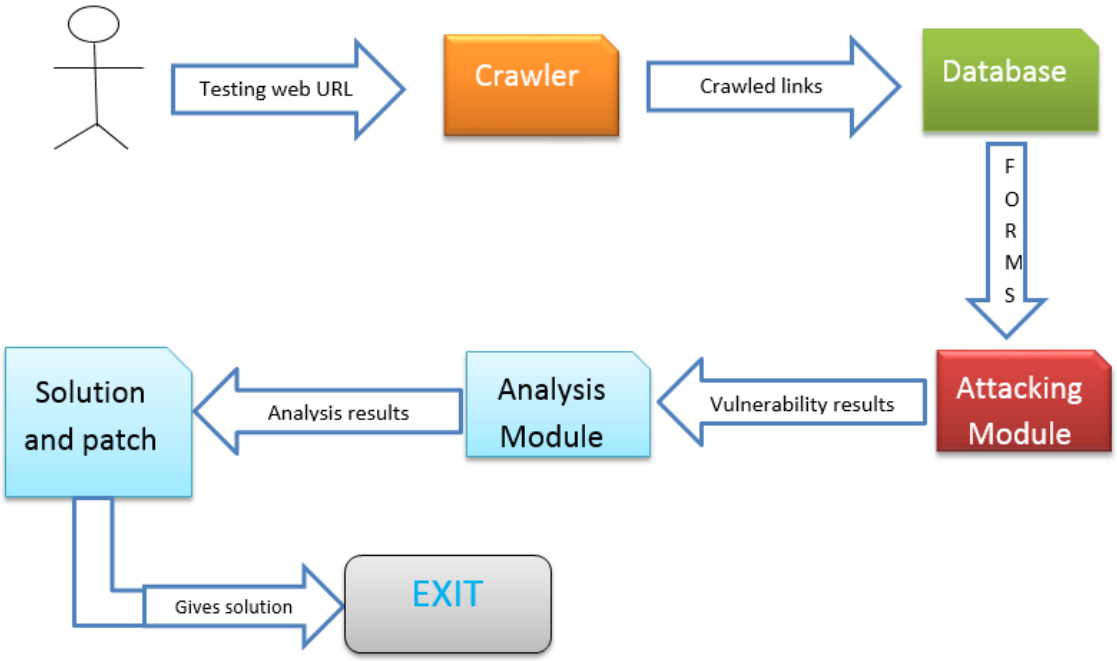
8. Portability:

- The application is easily portable, since the compatibility of the system requirements are met.

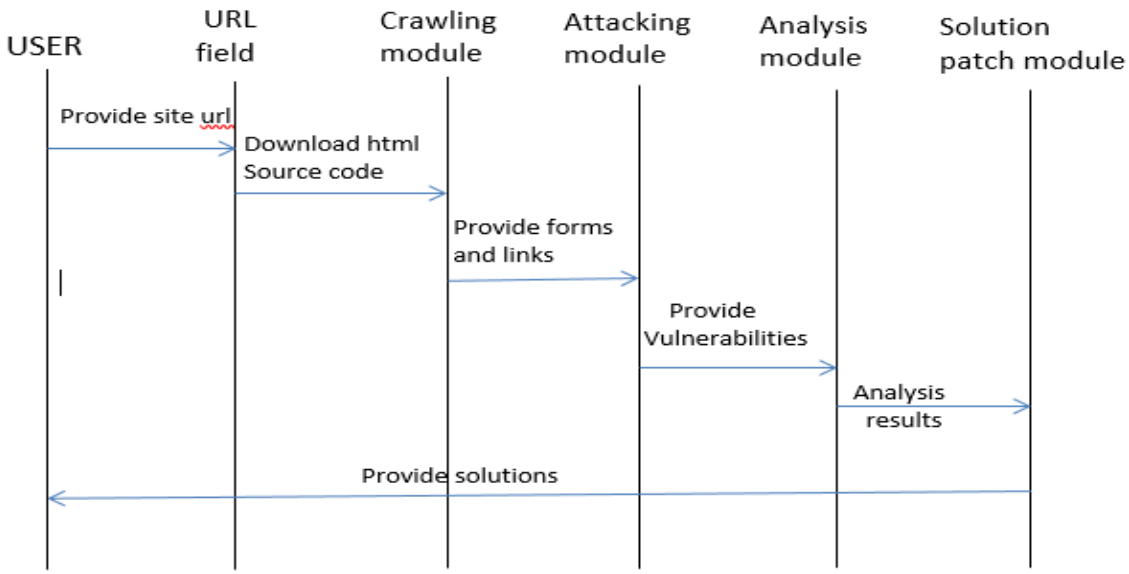
CHAPTER 8

UML DIAGRAMS

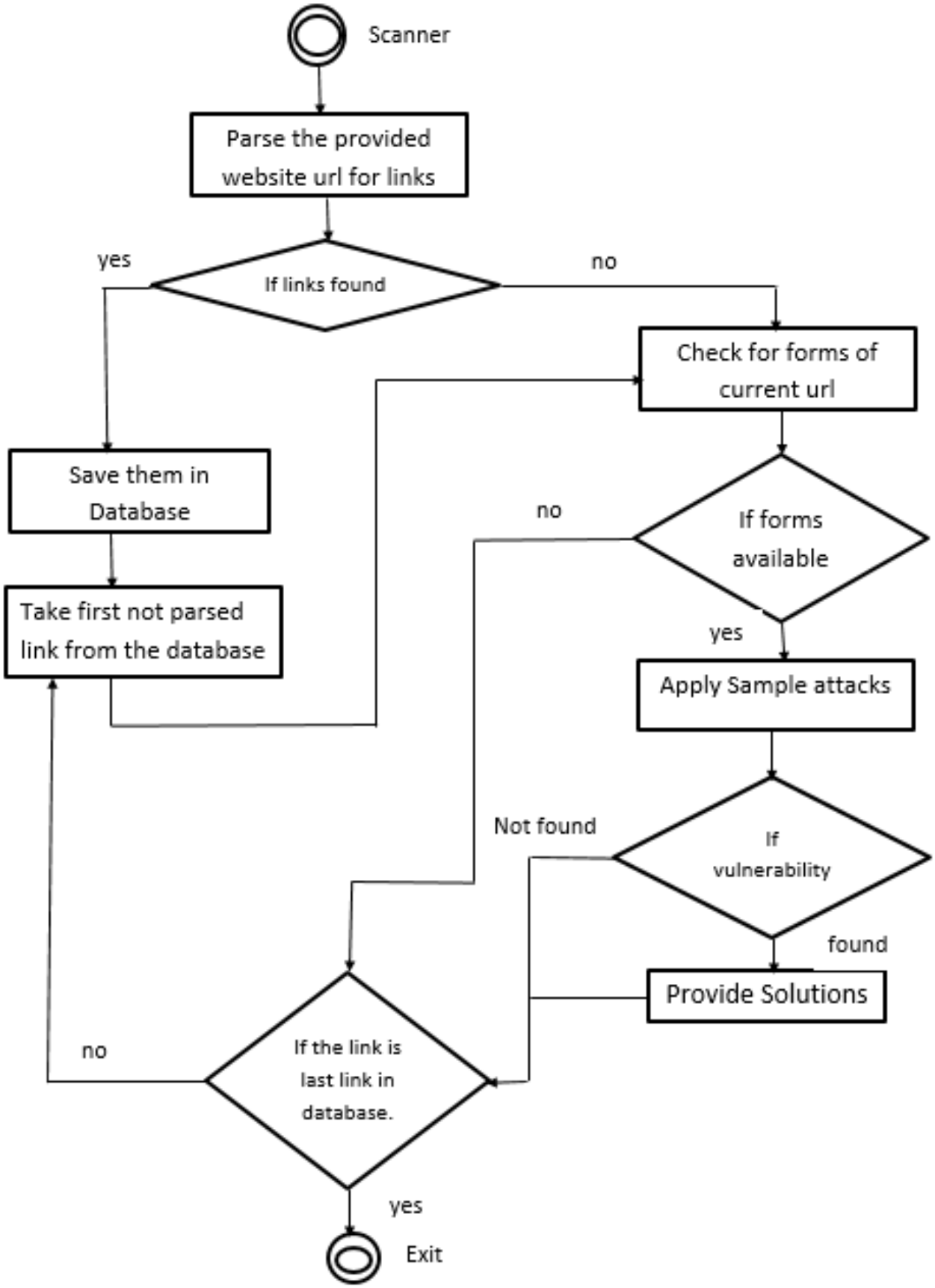
8.1. Data Flow Diagram



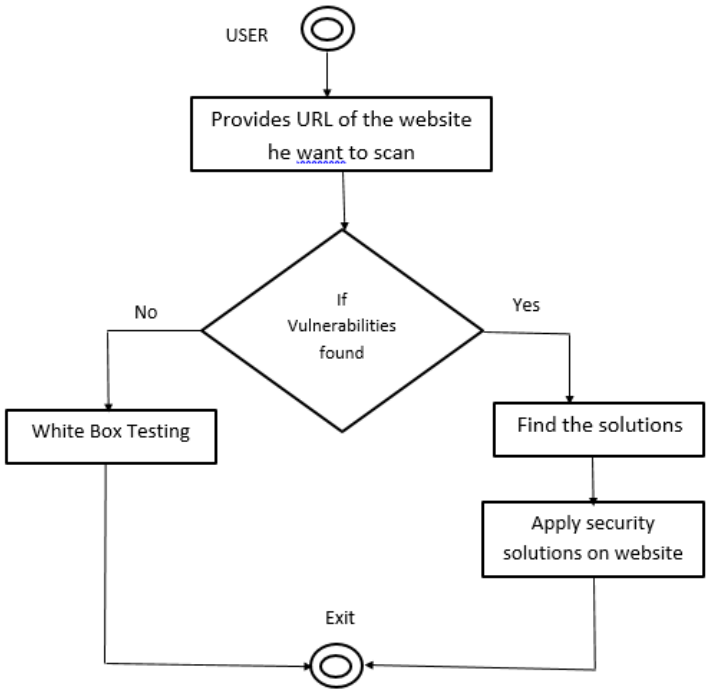
8.2. Sequence Diagram



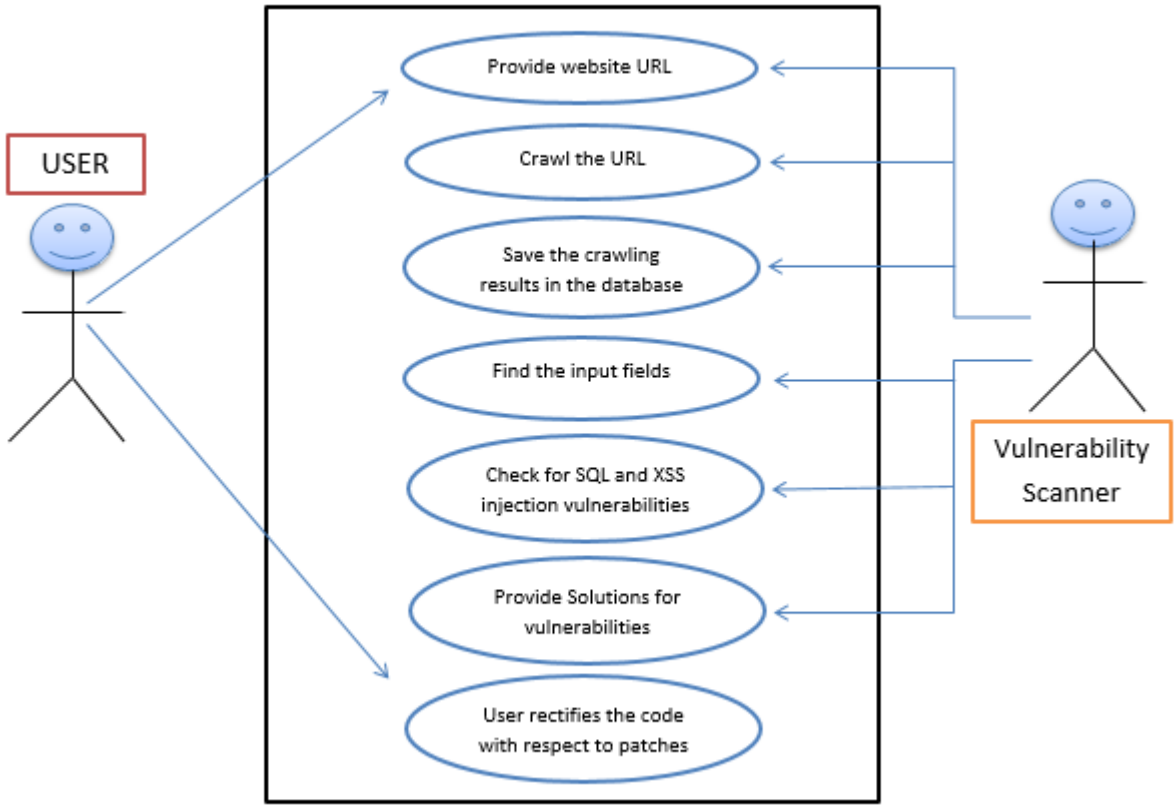
8.3. Scanner Activity Diagram



8.4. User Activity Diagram



8.5. Use Case Diagram



CHAPTER 9

OUR IMPLEMENTATION

9.1. Code for Crawling

```
import java.io.IOException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Attribute;
import org.jsoup.nodes.Attributes;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
public class Scanner {
    public static DB db = new DB();
    public static void main(String[] args) throws SQLException, IOException {
        db.runSql2("TRUNCATE Record;");
        String url = "http://10.1.11.111/onb";
        print("Fetching %s...", url);

        Document doc = Jsoup.connect(url).get();
        Elements links = doc.select("a[href]");
```

```
    processPage(url,doc);


    for (Element link : links) {
        processPage(link.attr("abs:href"),doc);
    }
    attacker();
}
public static void attacker()throws SQLException, IOException{
    String sql="select URL from Record";
    ResultSet rs= db.runSql(sql);
    String url=null;
    String z = null;
    String y = null;
    String temp=null;
    String temp2=null;
    while(rs.next())
    {
        url=rs.getString("URL");
        Document doc = Jsoup.connect(url).get();
        print("Finding forms of:- %s",url);
        Elements forms = doc.getElementsByTag("form");
        for(Element form : forms)
        {
            System.out.println(form.id());
            Attributes attributes = form.attributes();
```

```
for(Attribute attribute : attributes)
{
   System.out.println(attribute.toString());
   String x=attribute.toString();
   if(x.startsWith("action") || x.startsWith("ACTION"))
   {
      y= x.substring(8,x.length()-1);
      System.out.println(y);
   }
   if(x.startsWith("method") || x.startsWith("METHOD"))
   {
      z= x.substring(8,x.length()-1);
      System.out.println(z);
   }
}
Elements inputElements = form.getElementsByTag("input");
for (Element inputElement : inputElements) {
String key = inputElement.attr("name");
if("GET".equals(z) || "get".equals(z))
{
 temp=url+"/"+y+"?"+key+"=CHECKME";
 temp2=url+"/"+y+"?"+key+"=";
 sqllink sl =new sqllink(temp2);
}
else if("POST".equals(z) || "POST".equals(z))
```

```
            {
             temp=url+"/"+y+"?"+key+"=CHECKME";
             link2 li2 = new link2(temp);
             sqllink2 sl2 = new sqllink2(temp2);
            }
            }
          }
       }
   }

   private static void print(String msg, Object... args) {
      System.out.println(String.format(msg, args));
   }

   public static void processPage(String url,Document doc) throws SQLException, IOException{
      String sql = "select * from Record where URL = '"+url+"'";
      ResultSet rs = db.runSql(sql);
      if(rs.next()){
      }else{
         sql = "INSERT INTO `Crawler`.`Record` " + "(`URL`) VALUES " + "(?);";
         PreparedStatement stmt = db.conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
         stmt.setString(1, url);
         stmt.execute();
         System.out.println(url);
```

```
        }
        }
}
```

9.2. Code for SQL Injection Scanning

```
import urllib2
import time
import re
import mechanize


SQLeD = {'MySQL': 'error in your SQL syntax',
        'SQLi_err': 'access shop category information',
        'MySQL_fetch': 'mysql_fetch_assoc()',
        'MySQL_array': 'mysql_fetch_array()',
        'MySQL_result': 'mysql_free_result()',
        'MySQL_start': 'session_start()',
        'MYSQL': 'getimagesize()',
        'MySQL_call': 'Call to a member function',
        'Oracle1': 'Microsoft OLE DB Provider for Oracle',
        'Mysql_re': 'Warning: require()',
        'MysQLi': 'mysql_query()',
        'Oracle': 'ORA-01756',
        'MiscError': 'SQL Error',
        'MiscError2': 'mysql_fetch_row',
        'MiscError3': 'num_rows',
        'JDBC_CFM': 'Error Executing Database Query',
```

```
        'JDBC_CFM2': 'SQLServer JDBC Driver',
        'MSSQL_OLEdb': 'Mircrosoft OLE DB Provider for SQL Server',
        'MSSQL_Uqm': 'Unclosed quotation mark',
        'MS-Access_ODBC': 'ODBC Microsoft Access Driver',
        'Postgrey_error': 'An error occured',
        'SQL_errore': 'Unknown Column',
        'MS-Access_JETdb': 'Microsoft JET Database'}
print "\n[+]URL WITH PAYLOAD :",host
br=mechanize.Browser()
br.set_handle_robots(False)
br.set_handle_refresh(False)
response = br.open(host)
flist=list(br.forms())
for i in flist:
   br.form=i
   br.set_all_readonly(False)
   for control in br.form.controls:
      if(control.name!=None):
         try:
            br.form[control.name]="' or 1=1\""
         except:
            "Found Dynamic Content"
      else:
         try:
            control.value="' or 1=1\""
```

```
            except:
                "Found Dynamic Content"
        try:
            response=br.submit()
            r=response.read()
            #print r
            for type,eMSG in SQLeD.items():
                try:
                    if re.search(eMSG,r):
                        print "\n[+]SQL INJECTION BUG FOUND ERROR TYPE IS:", type
                        break
                    else:
                        print "\ntrying",type,": Found No Vulnerability"
                except:
                    "PLEASE START INTERNET"
        except:
            print "Found Dynamic Content"
        try:
            br.back()
        except:
            "Found Dynamic Content"
```

9.3. Code for XSS Scanning

```
#!/usr/bin/env python
from urlparse import urlparse, parse_qs
from HTMLParser import HTMLParser
```

```
from datetime import datetime

import urllib

import urllib2

import sys

import signal

import re



KEYWORD = "CHECKME"          #Must be plaintext word unlikely to appear on the page natively

RND_START = "JAKSDKSZA"      #Random string to append to check values. If getting errors, try changing or reducing length

RND_END = "KLASKSWLQ"

URL = ""

NUM_REFLECTIONS = 0          #Number of times the parameter value is displayed in the code.



VERBOSE = True



CURRENTLY_OPEN_TAGS = []     #Currently open is modified as the html is parsed

OPEN_TAG = ""                #Open is saved once xsscheckval is found

OPEN_EMPTY_TAG = ""

TAGS_TO_IGNORE = []          #These tags are normally empty <br/> or should be ignored because don't need to close them but sometimes, not coded properly <br> and missed by the parser.
```

```
OCCURENCE_NUM = 0

OCCURENCE_PARSED = 0


CHARS_TO_CHECK = {

    "\"": "Double quote",

    "'": "Single quote",

    "<": "Left angle bracket",

    ">": "Right angle bracket",

    "/": "Normal slash",

    "\\": "Backwards slash"

}


STRINGS_TO_CHECK = {

    "<script>": "<script> script tag is not escaped. There is a strong chance of XSS.",

    "<img />": "<img /> image tag is not escaped. This can be dangerous if additional characters are allowed (see above list).",

    "alert(1);": "alert(1); was not escaped. By itself, this is not too dangerous unless coupled with onmouseover and a double quote."

}


################################################################################
##########################################

# MAIN FUNCTION

################################################################################
##########################################
```

```
def main():
    try:
        #Print an intro comment


        print "\n[*] starting at " + str(datetime.time(datetime.now())) + "\n"


        #Parse the command line arguments
        #if (len(sys.argv) != 2 or KEYWORD not in sys.argv[1]):
        #       printout("fatal", "Invalid usage.\nUsage: python paramanalyze.py <FULL URL REPLACING PARAM TO CHECK WITH " + KEYWORD + ">\nExample: python paramanalyze.py \"http://site.com/?param=" + KEYWORD + "\"\n")
        global URL
        URL = link


        #Load the supplied URL to see if page is valid and can be successfully loaded
        init_resp = make_request(URL)
        printout("info", "URL was successfully loaded.")


        if(VERBOSE):
            printout("info", "Response size is: " + str(len(init_resp)) + " characters")


        #Now that the URL is valid, see if the response contains the check value (KEYWORD)
        if(VERBOSE):
            printout("info", "Checking if the response contains the test check value.")
```

```
    if(KEYWORD.lower() in init_resp.lower()):
        global NUM_REFLECTIONS
        NUM_REFLECTIONS = init_resp.lower().count(KEYWORD.lower())
        printout("info", "Check value was reflected in response " + str(NUM_REFLECTIONS) + " time(s).")
    else:
        printout("fatal", "Check value not in response. Nothing to test.")

    #Loop through and run tests for each occurence
    for i in range(NUM_REFLECTIONS):
        print "\n"
        printout("info", "Testing occurence number: " + str(i + 1))
        global OCCURENCE_NUM
        OCCURENCE_NUM = i+1
        test_occurence(init_resp)
        #Reset globals for next instance
        global CURRENTLY_OPEN_TAGS, OPEN_TAG, OCCURENCE_PARSED, OPEN_EMPTY_TAG
        CURRENTLY_OPEN_TAGS, OPEN_TAGS = [], []
        OCCURENCE_PARSED = 0
        OPEN_EMPTY_TAG = ""

    printout("exit", "Scan complete.")
  except KeyboardInterrupt:
```

```
        printout("exit", "Ctrl+C was pressed.")


#Try various tests against the specific occurence
def test_occurence(init_resp):
    #Begin parsing HTML tags to see where located
    parser = MyHTMLParser()
    location = ""
    try:
        parser.feed(init_resp)
    except Exception as e:
        location = str(e)
    except:
        printout("fatal", "Parsing error. Try rerunning?")


    if(location == "comment"):
        printout("info", "Parameter reflected in an HTML comment.")
    elif(location == "script_data"):
        printout("info", "Parameter reflected as data in a script tag.")
    elif(location == "html_data"):
        printout("info", "Parameter reflected as data or plaintext on the page.")
    elif(location == "start_end_tag_attr"):
        printout("info", "Parameter reflected as an attribute in an empty tag.")
    elif(location == "attr"):
```

```
    printout("info", "Parameter reflected as an attribute in an HTML tag.")
  else:
    printout("info", "Parameter is on the page but in an obscure location.")

  #Print full tag series leading up to location
  printout("info", "Open tag series preceeding parameter: " + str(CURRENTLY_OPEN_TAGS))
  printout("info", "Last opened tag: " + OPEN_TAG)

  #Begin scanning
  if(test_param_check(";!--\"<XSS>=&{()}") == ";!--\"<XSS>=&{()}"):
    printout("info", "Critical risk of XSS. ;!--\"<XSS>=&{()} reflected.")
  else:
    printout("info", "Some filtering is being done. Investigating...")

    for key, value in CHARS_TO_CHECK.iteritems():
      #print item + " and " + CHARS_TO_CHECK[item]
      char_to_check_resp = test_param_check(key)
      if(char_to_check_resp == key):
        printout("warn", value + " is not escaped: " + key)
      else:
        printout("info", value + " is escaped as: " + char_to_check_resp)

    for key, value in STRINGS_TO_CHECK.iteritems():
```

```
        string_to_check_resp = test_param_check(key)
        if(string_to_check_resp == key):
            printout("warn", value)
        else:
            printout("info", key + " is escaped as: " + string_to_check_resp)
```

#Tests to see if the param_to_check is reflected in the code as param_to_compare. Can be same or urlescaped.

```
def test_param_check(param_to_check):
    check_string = RND_START + param_to_check + RND_END
    check_url = URL.replace(KEYWORD, urllib.quote_plus(check_string))
    try:
        check_response = make_request(check_url)
    except:
        check_response = ""
    response = ""

    #Loop to get to right occurence
    occurence_counter = 0
    for m in re.finditer(RND_START, check_response, re.IGNORECASE):
        occurence_counter += 1
        if(occurence_counter == OCCURENCE_NUM):
        #if((occurence_counter == OCCURENCE_NUM) and (check_response[m.start():m.start()+len(compare_string)].lower() == compare_string.lower())):
```

```
            remaining_str = check_response[m.start():]
            #print remaining_str
            rnd_end_pos = remaining_str.index(RND_END)
            response = remaining_str[len(RND_START):rnd_end_pos]
            break
    return response


#Makes a request to the supplied in_url and returns the response.
def make_request(in_url):
    try:
        req = urllib2.Request(in_url)
        resp = urllib2.urlopen(req)
        return resp.read()
    except:
        printout("fatal", "Could not load the URL. The network could be down or the page may be returning a 404.")


#Prints a message with [INFO] or [WARN] in front along with the current time
def printout(msg_type, msg):
    if(msg_type == "info"):
        print "[" + str(datetime.time(datetime.now())) + "]" + " [INFO] " + msg
    elif(msg_type == "warn"):
        print "[" + str(datetime.time(datetime.now())) + "]" + " [WARNING] " + msg
    elif(msg_type == "fatal"):
```

```
        print "[" + str(datetime.time(datetime.now())) + "]" + " [FATAL] " + msg
        print "\n[*] shutting down at " + str(datetime.time(datetime.now())) + "\n"
        exit()
    elif(msg_type == "exit"):
        print "[" + str(datetime.time(datetime.now())) + "]" + " [INFO] " + msg
        print "\n[*] shutting down at " + str(datetime.time(datetime.now())) + "\n"
        exit()
#HTML Parser class
class MyHTMLParser(HTMLParser):
    def handle_comment(self, data):
        global OCCURENCE_PARSED
        if(KEYWORD.lower() in data.lower()):
            OCCURENCE_PARSED += 1
            if(OCCURENCE_PARSED == OCCURENCE_NUM):
                raise Exception("comment")

    def handle_startendtag(self, tag, attrs):
        global OCCURENCE_PARSED
        global OCCURENCE_NUM
        global OPEN_EMPTY_TAG
        global OPEN_TAG
        if (KEYWORD.lower() in str(attrs).lower()):
            OCCURENCE_PARSED += 1
```

```
        if(OCCURENCE_PARSED == OCCURENCE_NUM):
            OPEN_EMPTY_TAG = tag
            OPEN_TAG = tag
            raise Exception("start_end_tag_attr")
    def handle_starttag(self, tag, attrs):
        global CURRENTLY_OPEN_TAGS
        global OPEN_TAG
        global OCCURENCE_PARSED
        #print CURRENTLY_OPEN_TAGS
        if(tag not in TAGS_TO_IGNORE):
            CURRENTLY_OPEN_TAGS.append(tag)
            OPEN_TAG = tag
        if (KEYWORD.lower() in str(attrs).lower()):
            if(tag == "script"):
                OCCURENCE_PARSED += 1
                if(OCCURENCE_PARSED == OCCURENCE_NUM):
                    raise Exception("script")
            else:
                OCCURENCE_PARSED += 1
                if(OCCURENCE_PARSED == OCCURENCE_NUM):
                    raise Exception("attr")
    def handle_endtag(self, tag):
        global CURRENTLY_OPEN_TAGS
```

```
        global OPEN_TAG
        global OCCURENCE_PARSED
        if(tag not in TAGS_TO_IGNORE):
            CURRENTLY_OPEN_TAGS.remove(tag)
    def handle_data(self, data):
        global OCCURENCE_PARSED
        if (KEYWORD.lower() in data.lower()):
            OCCURENCE_PARSED += 1
            if(OCCURENCE_PARSED == OCCURENCE_NUM):
                #If last opened tag is a script, send back script_data
                #Try/catch is needed in case there are no currently open tags, if not, it's considered data (may occur with invalid html when only param is on page)
                try:
                    if(CURRENTLY_OPEN_TAGS[len(CURRENTLY_OPEN_TAGS)-1] == "script"):
                        raise Exception("script_data")
                    else:
                        raise Exception("html_data")
                except:
                    raise Exception("html_data")
#RUN MAIN FUNCTION
if __name__ == "__main__":
    main()
```

CHAPTER 10

EXPERIMENTAL RESULTS

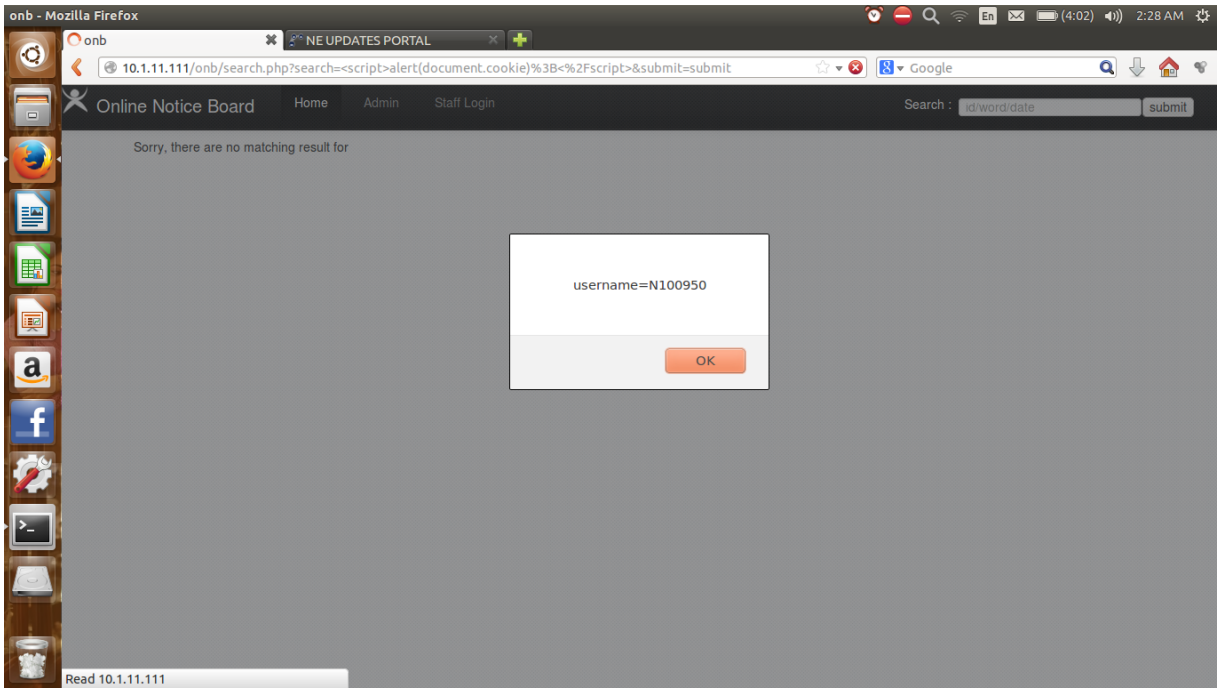
10.1. Scanning results of RGUKT ONB

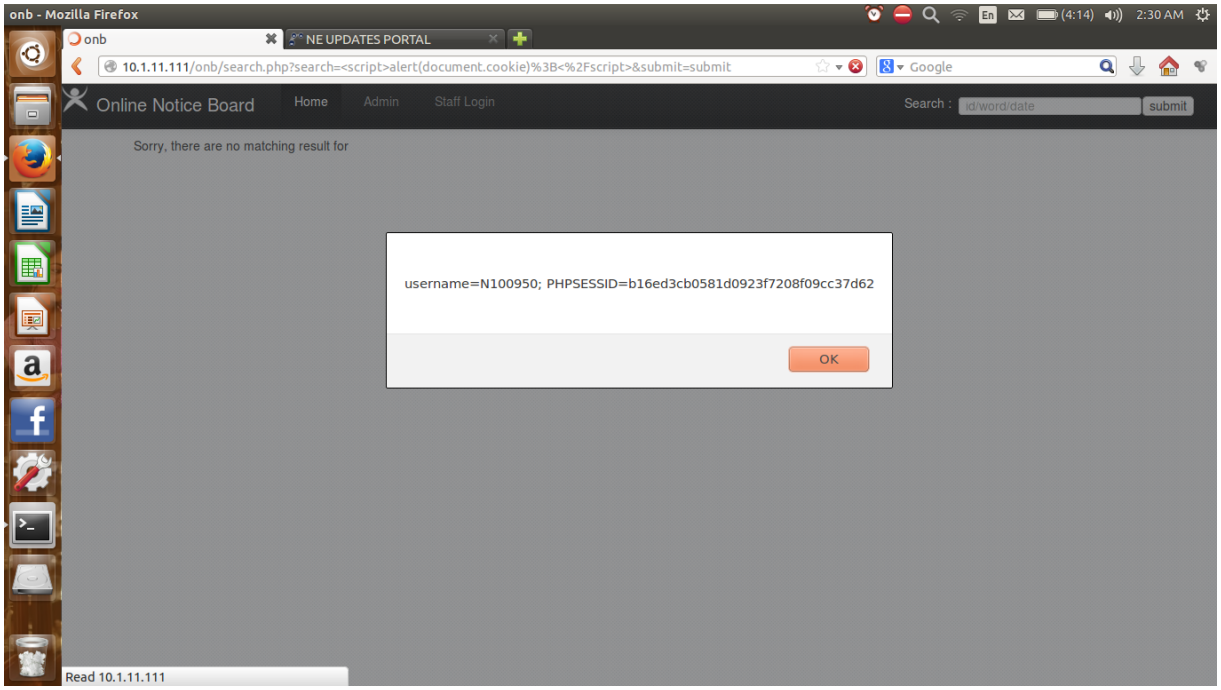
For testing our application working nature, we targeted for the vulnerabilities of RGUKT ONB. The scanner successfully crawled the website and found xss vulnerability.

Vulnerability result:- <script> tag was not escaped.

For confirming the successful working of scanner we implemented a sample attack manually on ONB.

We submitted search field which was found by crawler in the scanner with the following input <script>alert(document.cookie)</script>. Search field must not consider this as normal search input, but the ONB showing an alert box with SessionId. With this SessionId any untrusted user can login to the ONB. This process is called session hijacking.

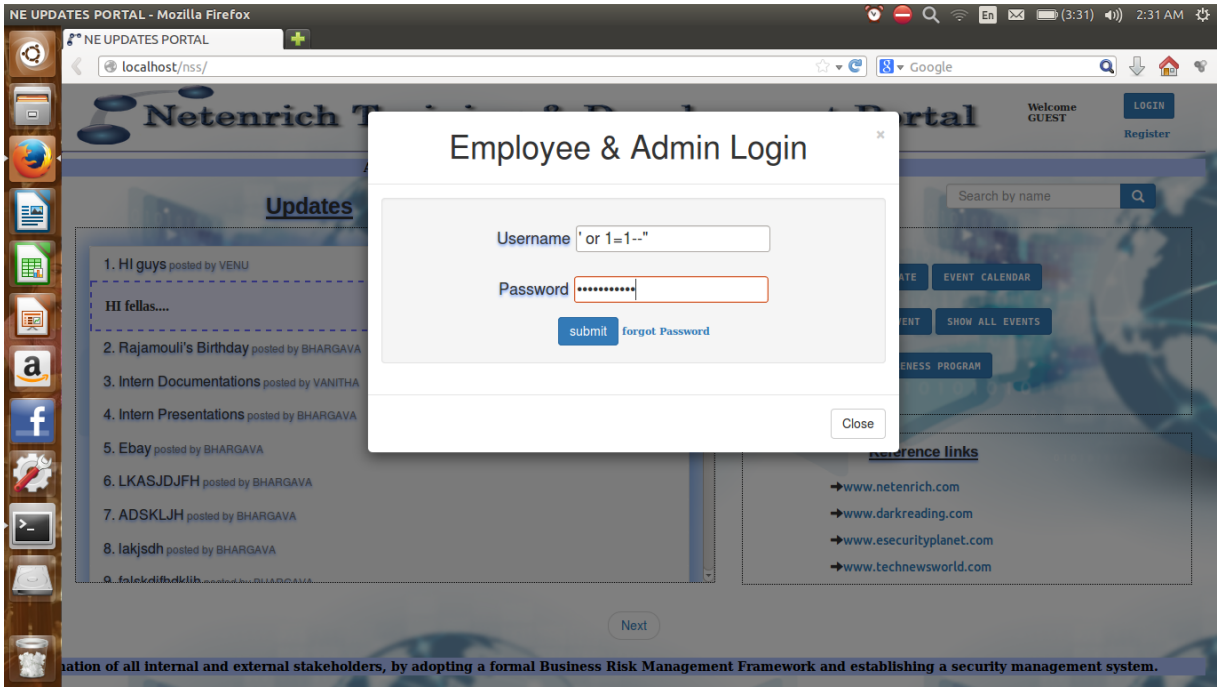


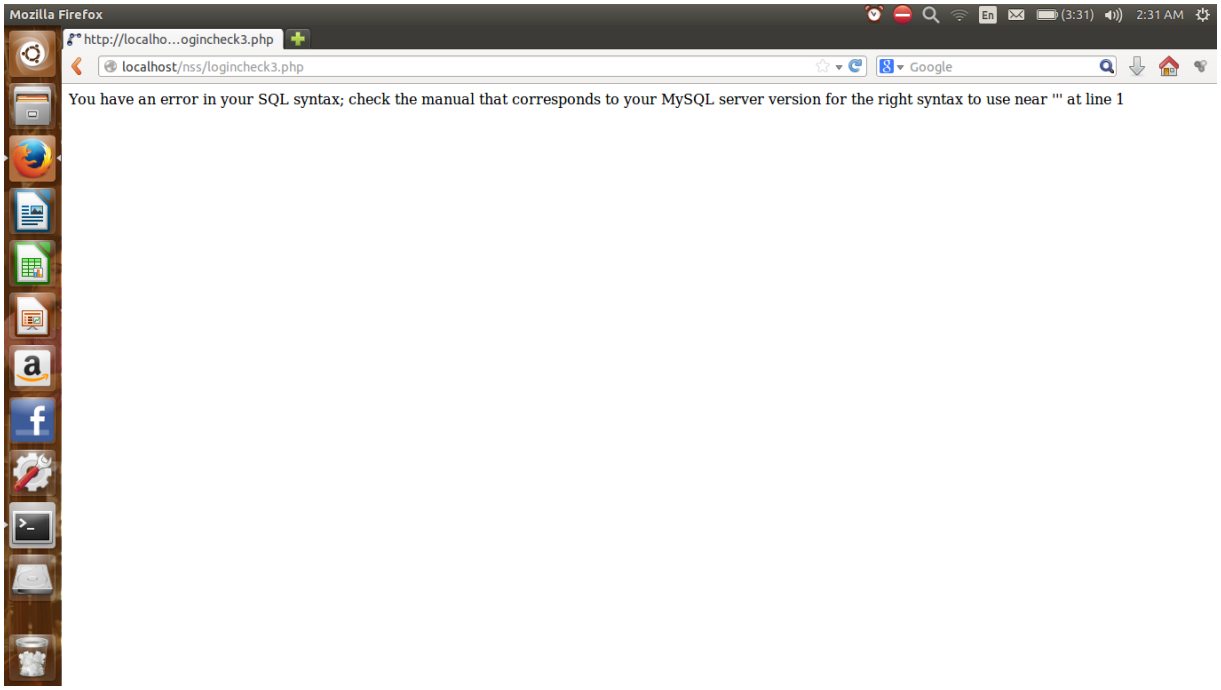


10.2. Scanning results of NSS Update Portal

We targeted another website that is RSS Update Portal to find vulnerabilities. We found that there is a possibility of sql injection vulnerability through login form.

Vulnerability Result: Found sql error.




Mozilla Firefox
(3:31)
2:31 AM
http://localho...ogincheck3.php
localhost/nss/logincheck3.php
Google
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

CHAPTER 11

LIMITATIONS OF SCANNER

- Vulnerability scanners can only report vulnerabilities according to the sample attacks integrated in the attacking module. Human judgment is always needed in analyzing the data after the scanning process.
- Some of the websites developed dynamic web pages. Crawler cannot parse the dynamic content.
- Urls redirecting to other websites could be problem while scanning.
- Scanner will use settings of default browser of the current operating system. So, it is required to user must keep the browser setting as default.
- Proxy authentication is not developed with scanner. So the testing url must be accessible without the proxy authentication.

CHAPTER 12

CONCLUSION AND FUTURE SCOPE

12.1. Conclusion

This reports present current working nature of the vulnerability scanner. We implemented possible attacking techniques in the attacking module. There is a chance that new technique may arise in future for vulnerability exploitation. We need to maintain the scanner like we have to update attacking module frequently.

12.2. Future Scope

Current vulnerability scanner only check for sql and xss vulnerabilities. In future we can develop and include owasp remaining top 8 vulnerability checking processes in the application. We can implement deep web crawling techniques with the scanner in future.

REFERENCES

[1] https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

[2] https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

[3] https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[4] https://www.owasp.org/index.php/Top_10_2013-Top_10

[5] https://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf