ScanF: Web Application Security Assessment by Fault Injection and Behavior Monitoring

Software Project lab-3

NOVEMBER 25, 2018

INSTITUTE OF INFORMATION TECHNOLOGY

University of Dhaka

ScanF: Web Application Security Assessment by Fault Injection and Behavior Monitoring

Supervised by

B M Mainul Hossain

Associate Professor

Institute of Information Technology

University of Dhaka

Submitted by

Rafed Muhammad Yasir

(BSSE 0733)

BSSE Session: 2014-2015

Institute of Information Technology

University of Dhaka

Institute of Information Technology

University of Dhaka

25-11-2018

Abstract

This document is a technical report of the system to be built- ScanF: web application security assessment by fault Injection and behavior monitoring. The tool aims to detect SQL vulnerabilities and cross site scripting vulnerabilities using fault injection methods. This document contains the software requirement specifications, design specifications, implementation details, testing procedures that were followed to build the system. Reading this document and following the user manual will provide the reader a general idea of the structure of the system and how to use it.

Letter of Transmittal

November 25, 2018

BSSE 4th Year Exam Committee

Institute of Information Technology

University of Dhaka

Dear Sir,

I have prepared the report on “ScanF: A web application security assessment tool by fault injection and behavior monitoring”. This report includes the details of each steps I followed from gathering requirements to implementing the tool.

The primary purpose of this report is to summarize my working procedure for building the tool so that anyone reading this document can understand how ScanF works and use it as well to its full potential for scanning vulnerabilities in websites. I have tried my level best to implement the tool in the best way possible.

Sincerely yours,

Rafed Muhammad Yasir

BSSE 0733

Document Authentication

This project document has been approved by the following persons.

--------------------------------	-------------------------------
Prepared by	**Approved by**
Rafed Muhammad Yasir	B M Mainul Hossain
BSSE-0733	Associate Professor
	Institute of Information Technology
	University of Dhaka

Letter of Endorsement

This letter is to certify that, Rafed Muhammad Yasir, BSSE0733, student of Institute of Information Technology, University of Dhaka, has done "ScanF: A web application security assessment tool by fault injection and behavior monitoring". I have gone through the report. All the information mentioned in this document is true. I wish him every success in life and hope that he will continue his effort.

SPL Supervisor

Dr. B M Mainul Hossain
Associate Professor
Institute of Information Technology
University of Dhaka

Table of Contents

List of Figures

1. Introduction

Web applications nowadays are very matured and powerful. However, many web applications go through rapid development phases with very tight deadlines, making it difficult to eliminate all vulnerabilities.

The aim of this project is to develop a tool that will assess web application security based on fault injection techniques. A user will provide the tool with an application URL. The tool will automatically crawl the application and search for input fields where faults can be injected. Assessing the nature of the input field, faults will be injected in the system. After a fault has been injected, the tool will assess the behavior of the system. Any abnormal behavior will conclude that a successful injection has occurred and the system is vulnerable. The user will be able to observe and monitor how the tool is assessing the application through a user friendly interface. The user can also tweak strategies for different injection methods for better vulnerability discovery.

This document contains the software requirement specifications and design specifications of the tool to be built which will give the readers an idea of the entire system.

2. Quality Function Deployment (QFD)

Quality function deployment is a quality management technique that translates the needs of the customer into technical requirements for software. The following requirements for ScanF have been identified.

Normal requirements:

- The tool can identify forms and through which data is submitted
- The tool can run automated sql injections on identified form fields
- The tool can run automated JS injections for reflected XSS discovery

Expected requirements:

- The tool can crawl a website and list its pages
- Users can choose the type of payloads to be used for attacks
- Results of previously performed attacks will be stored and can be viewed
- Users can manually craft sql injection payloads for better inspection
- For better understanding of injection results, screenshots will be taken after an injection is performed

3. Usage Scenario

Crawling

To use the tool, a user will have to provide URL of an application he wants to test. The tool will start crawling for other pages present in the application. For better spidering of the application, the user can provide a session cookie returned by an application after authentication. After crawling of a page, the tool will search for data entry points where user can submit data. After a crawling session ends, the user will have a list of data entry points where faults can be injected. Crawling can be stopped midway and restarted again if needed.

SQL Injection

There will be two methods of SQL injection- automated and manual. In automated injections, the tool will test a form by injecting only one field at a time. A user will have several options to choose from on what SQL faults can be injected. The other fields in the form will be filled smartly and according to the constraints of each fields. The smart filling of other fields will increase the chances of a successful injection. All possible combinations of the attack will be carried out. Upon submitting each form, the response will be checked for signs of a successful injection.

In manual testing, the user can provide his own inputs that he finds suitable for testing. In this case, unlike automated testing, only one attack will be carried out at a time.

JS Injection

JavaScript injection will be performed to detect reflected cross site scripting (XSS). The tool will look at the parameters of a typical web application request and observe the response to see if any of the parameters are present (as a reflection). If reflected, the tool verifies that the reflection is not a coincidence.

Test result management

The result of each test will form a list which will be displayed to the user. Upon clicking a list, the details of the test performed will be shown. Along with textual test results, screenshot of the resultant page will be shown. Test items from the list can be deleted by a user.

4. Scenario Based Modeling

Although the success of a computer-based system or product is measured in many ways, user satisfaction resides at the top of the list. If we understand how end users (and other actors) want to interact with a system, a developer can characterize requirements and build meaningful analysis and design models. Hence, requirements modeling with begins with the creation of scenarios in the form of use cases and activity diagrams.

4.1. Use case diagrams

Use case diagrams of ScanF are given below. The system has only one actor (User). Thus, mention of actors in diagram description is omitted.

Level 0: ScanF

Goal in context: The diagram represents the whole ScanF System.

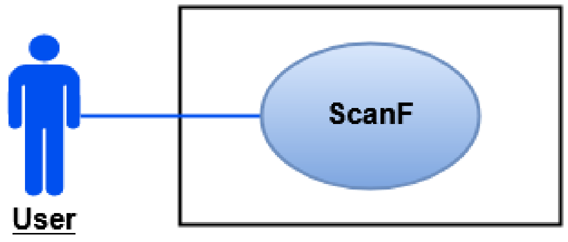


Figure 1: Level 0 use case – ScanF

Level 1: Modules of ScanF

Goal in context: The diagram shows all the modules of ScanF

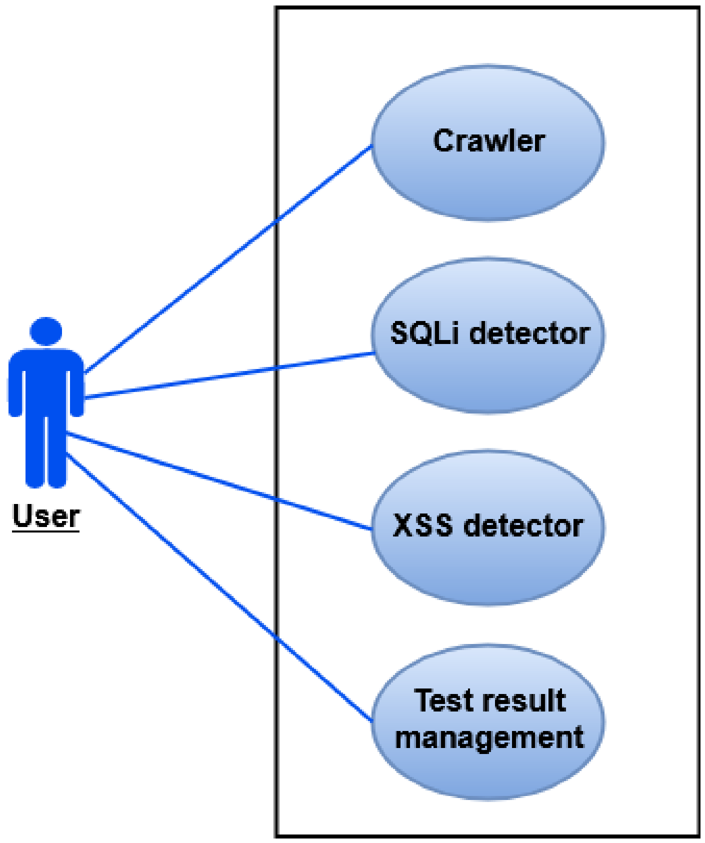


Figure 2: Level 1 use case -modules of ScanF

Level 1.1: Crawler

Goal in context: The diagram refers to the details of Crawler module of level 1.

Actions and Replies:

A1: User provides URL as input

R1: System crawls the site and finds forms

A2: User pauses/resumes crawling

R2: Crawler pauses/resumes

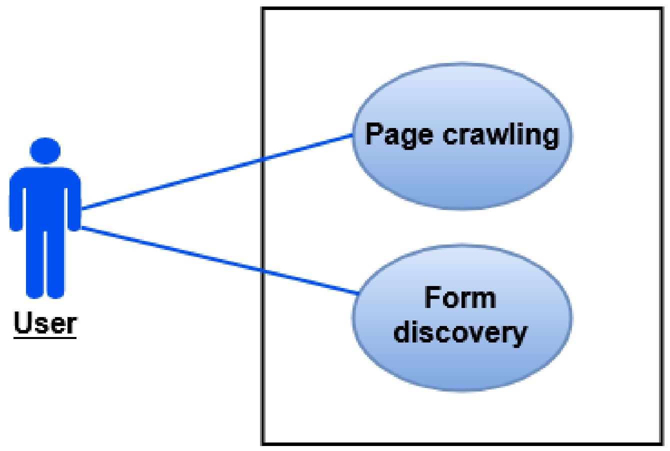


Figure 3: Level 1.1 use case – Crawler

Level 1.2: SQLi detector

Goal in context: The diagram refers to the details of SQLi detector module of level 1.

Actions and Replies:

A1: User initiates testing

R1: System injects forms and analyzes behavior

A2: User manually performs attack

R2: System carries out manual injection

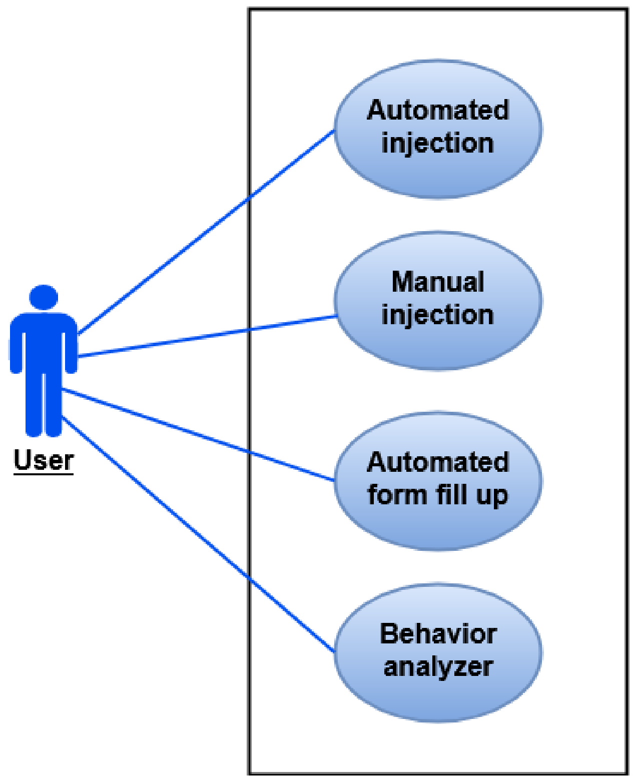


Figure 4: Level 1.2 use case - SQLi detector

Level 1.3: XSS detector

Goal in context: The diagram refers to the details of XSS detector module of level 1.

Actions and Replies:

A1: User initiates testing

R1: System injects forms and analyzes behavior

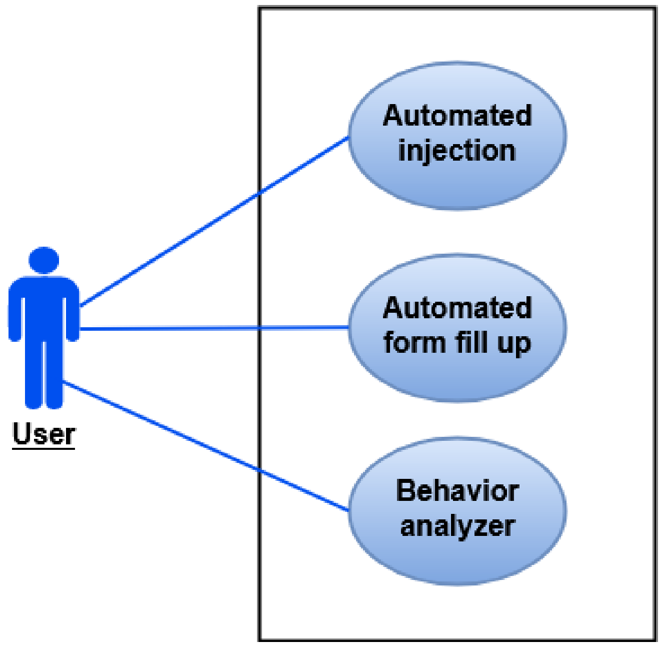


Figure 5: Level 1.3 use case - XSS detector

Level 1.4: Test result management

Goal in context: The diagram refers to the details of test result management module of level 1.

Action and Replies:

A1: User selects attack list

R1: List of attacks and their results are showed

A2: An attack is performed

R2: System takes screenshot of the response

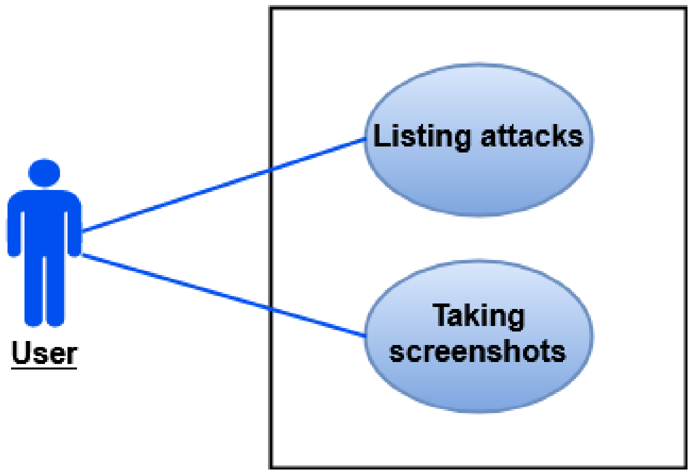


Figure 6: Level 1.4 use case - Test result management

4.2. Activity Diagrams

Activity diagrams are graphical representations of workflows of stepwise activities and actions. Activity diagram describes parallel and conditional activities, use cases and system functions at a detailed level [4].

Activity Diagrams of ScanF are given below.

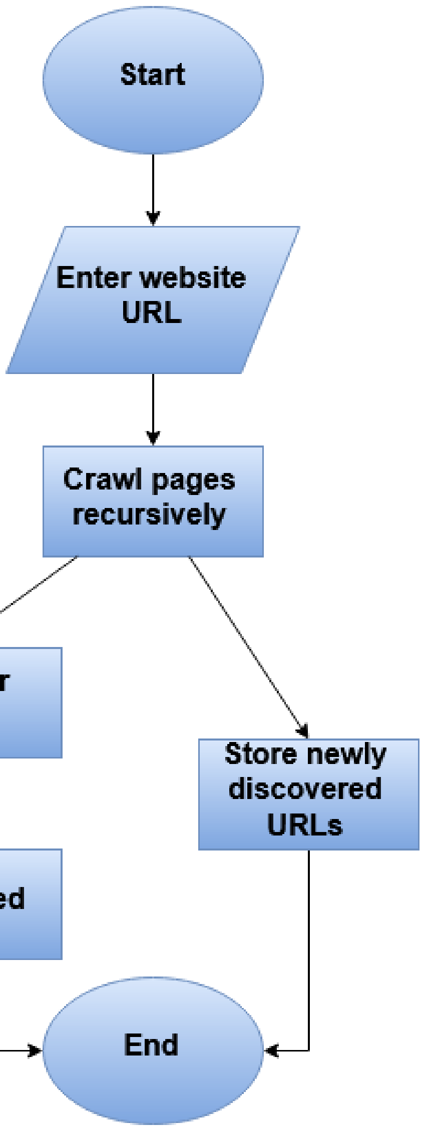


Figure 7: Activity diagram of crawling (from use case 1.1)

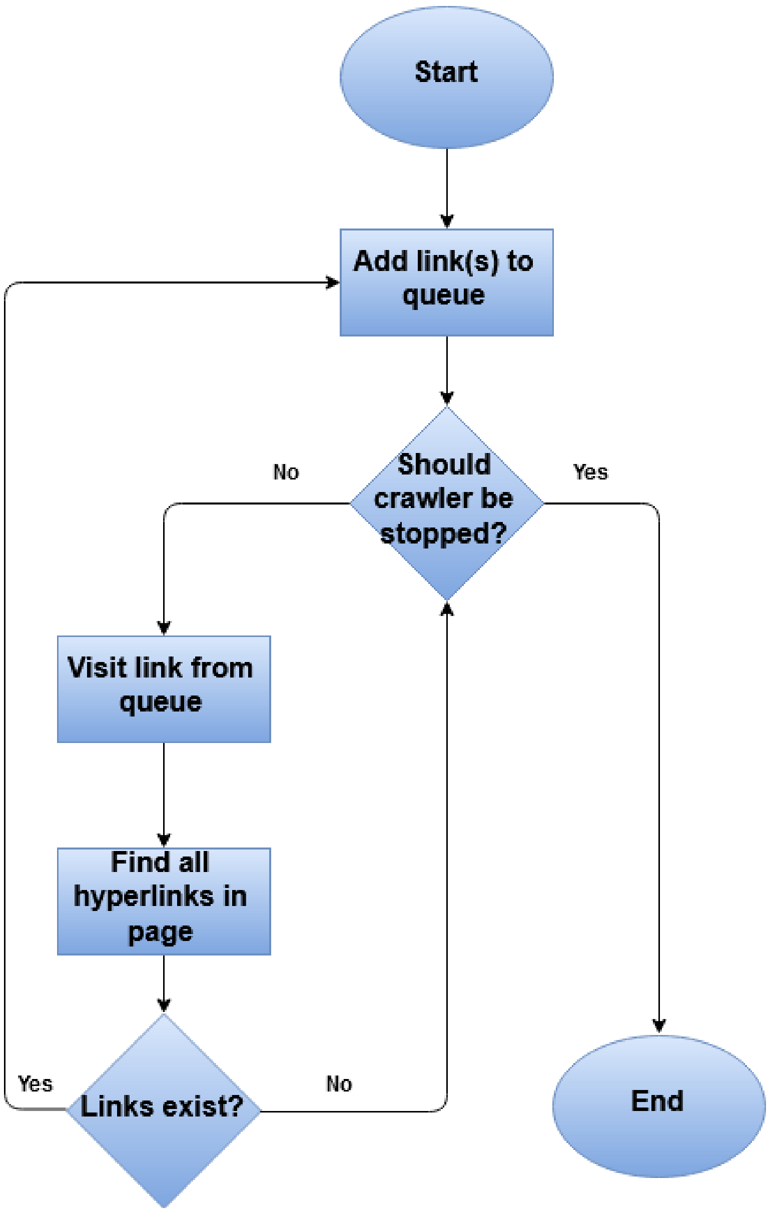


Figure 8: Activity diagram of page crawling (from use case 1.1.1)

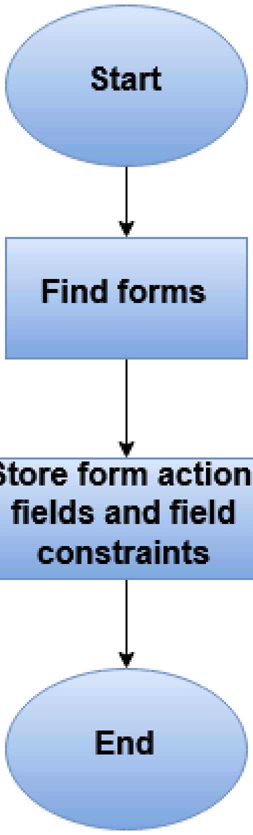


Figure 9: Activity diagram of form discovery (from user case 1.1.2)

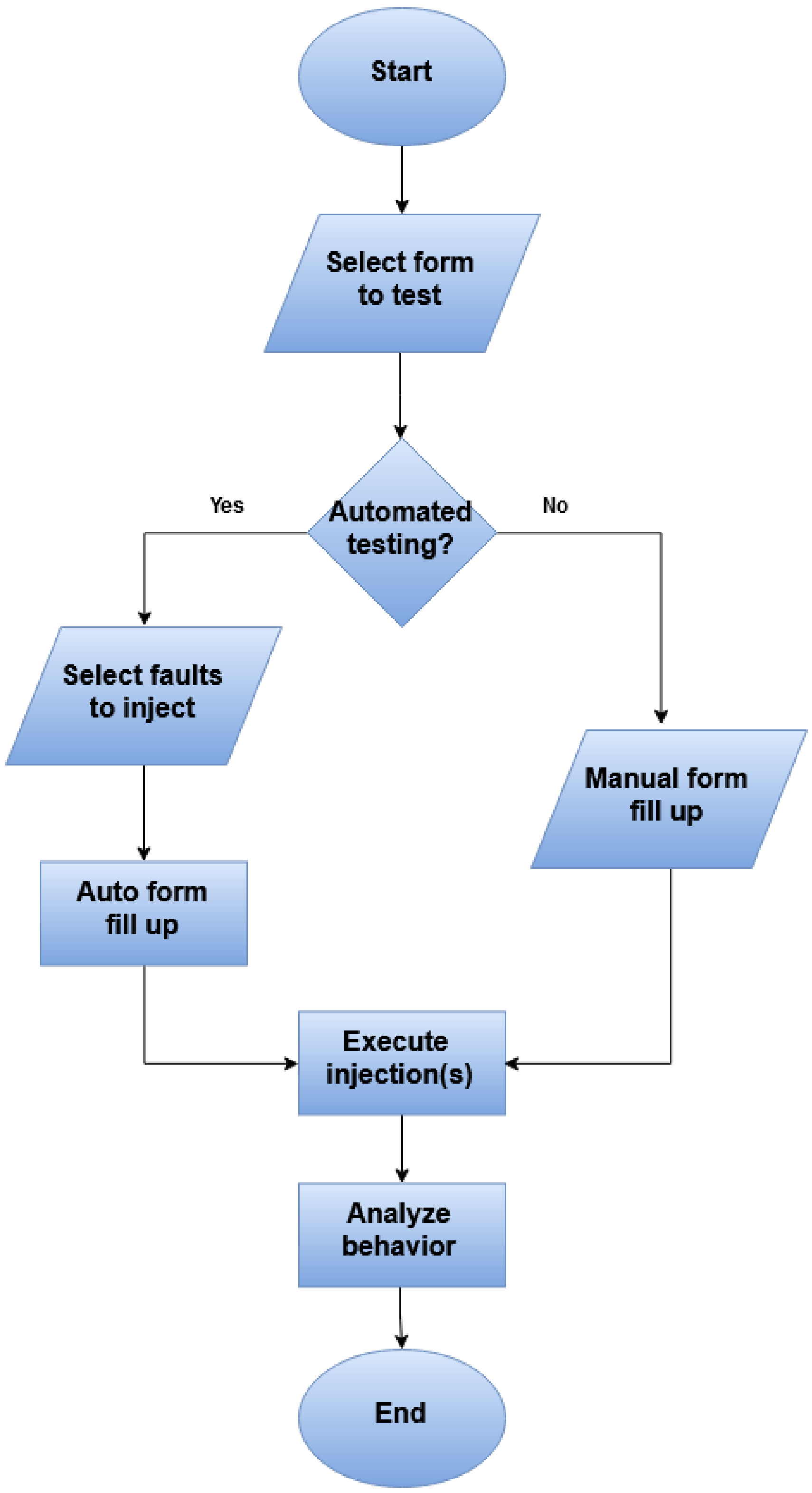


Figure 10: Activity diagram of SQLi detection (from use case 1.2)

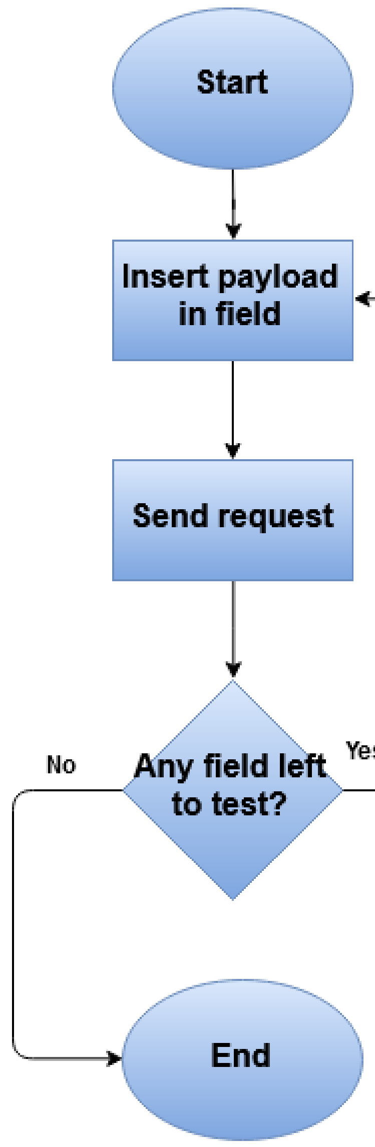


Figure 11: Activity diagram of automated injection (from use case 1.2.1)

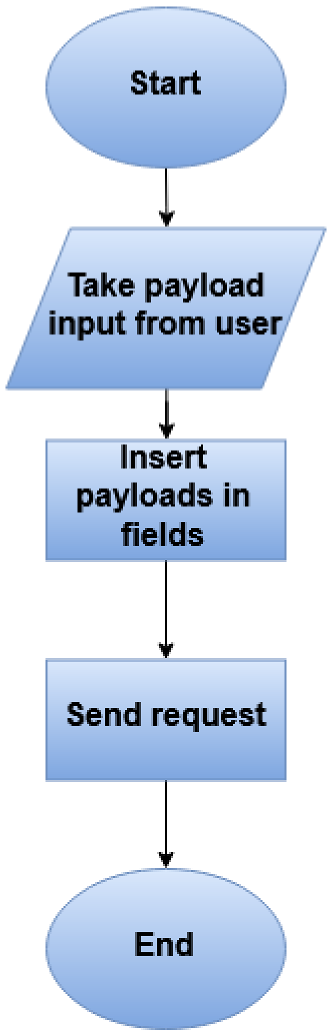


Figure 12: Activity diagram of manual injection (from use case 1.2.2)

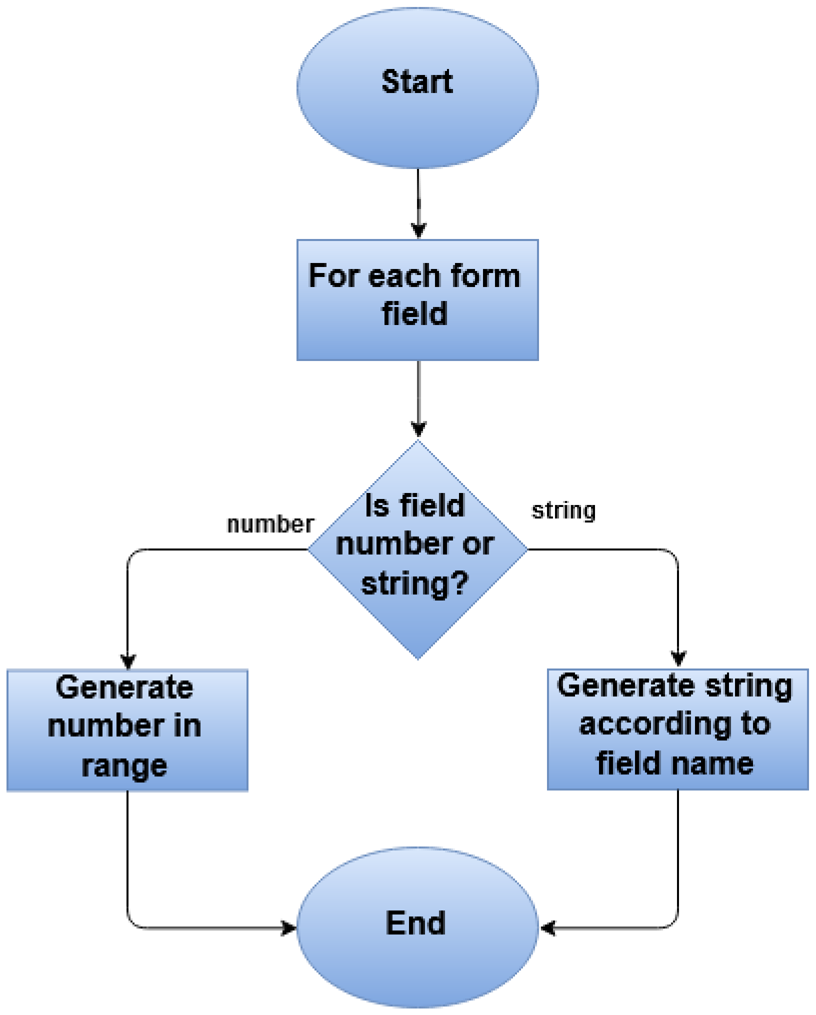

Start
For each form field
Is field number or string?
number
string
Generate number in range
Generate string according to field name
End

Figure 13: Activity diagram of automated form fill up (from use case 1.2.3)

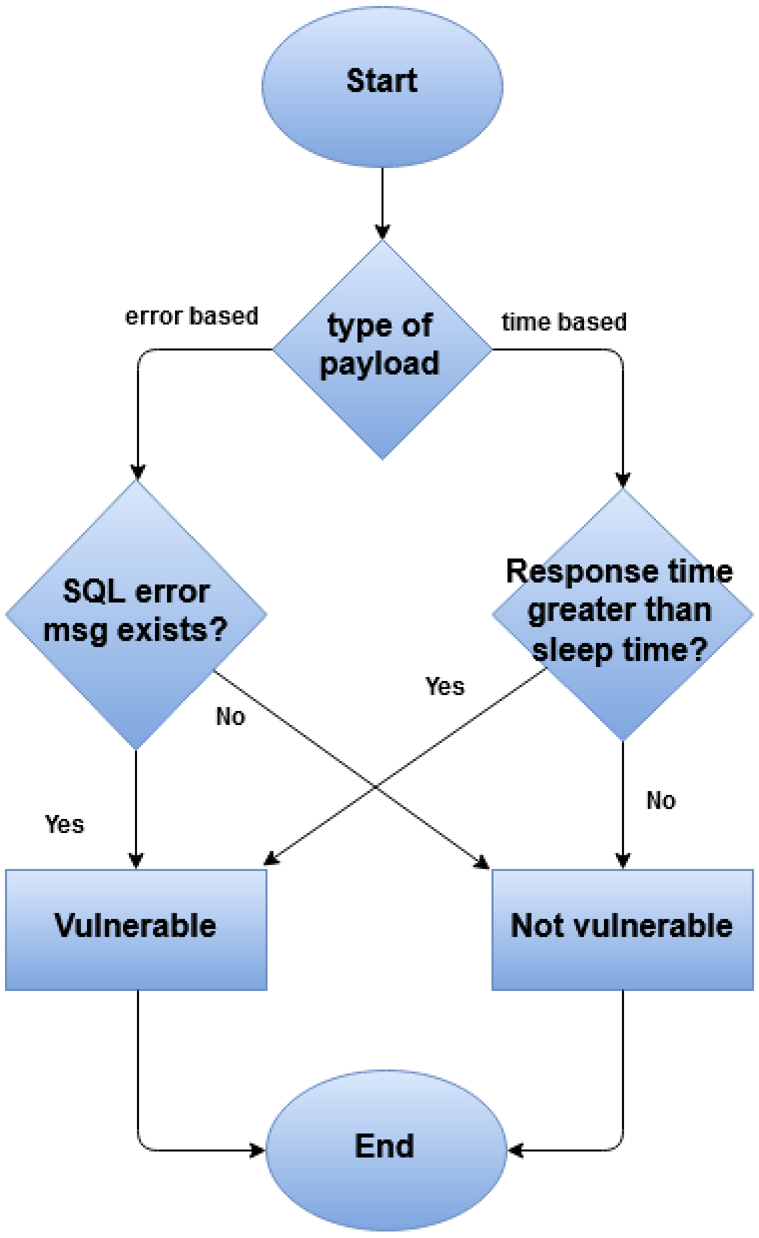


Figure 14: Activity diagram of behavior analyzer (from use case 1.2.4)

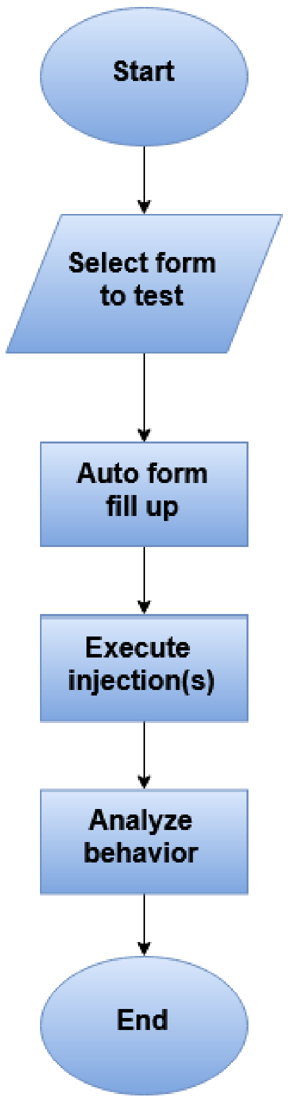


Figure 15: Activity diagram of XSS detection (from use case 1.3)

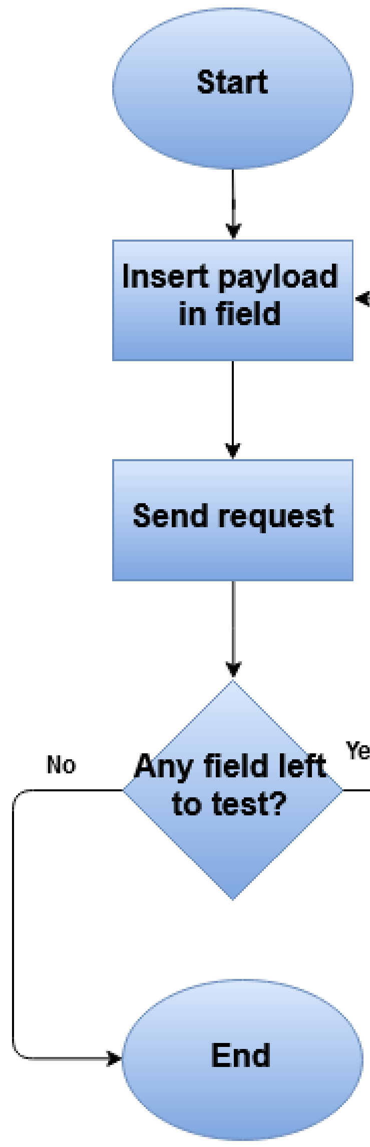


Figure 16: Activity diagram of automated injection (from use case 1.3.1)

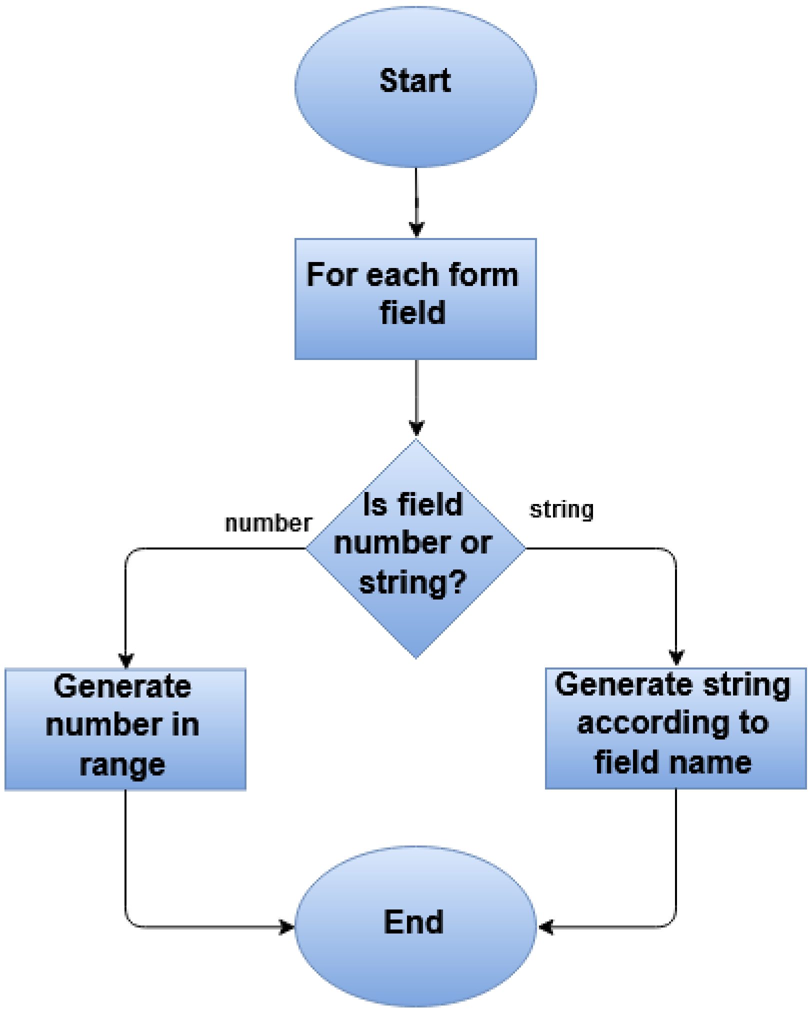

Start
For each form field
Is field number or string?
number
string
Generate number in range
Generate string according to field name
End

Figure 17: Activity diagram of automated form fill up (from use case 1.3.2)

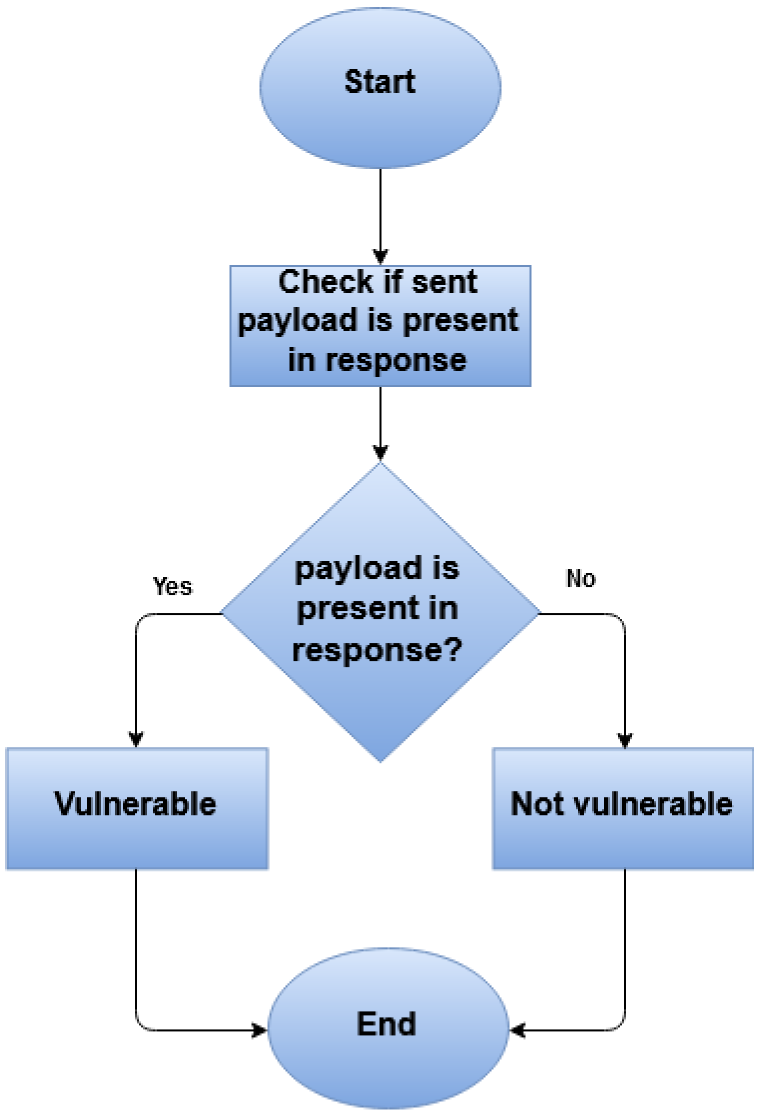

Start
Check if sent payload is present in response
payload is present in response?
Yes
No
Vulnerable
Not vulnerable
End

Figure 18: Activity diagram of behavior analyzer (from use case 1.3.3)

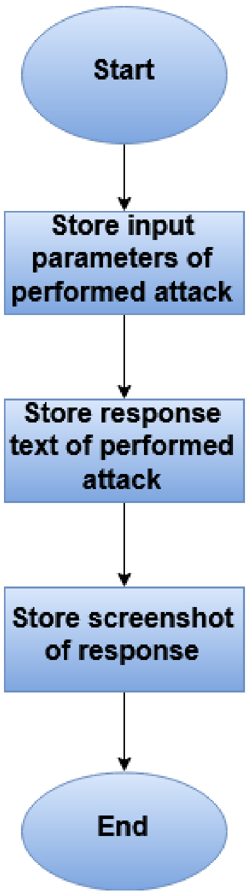

Start
Store input parameters of performed attack
Store response text of performed attack
Store screenshot of response
End

Figure 19: Activity diagram of test result management (from use case 1.4)

5. Data Modeling

Data modeling provides a description of how data are represented and accessed [4]. Data modeling includes:

- Identifying all data objects (a data object is a representation of composite information (something that has a number of different attributes) that must be understood by software) that are processed within the system.
- Identifying the relationships between the data objects and other information that is pertinent to the relationships.

After thorough analysis of the system the following data objects and their relationships were identified.

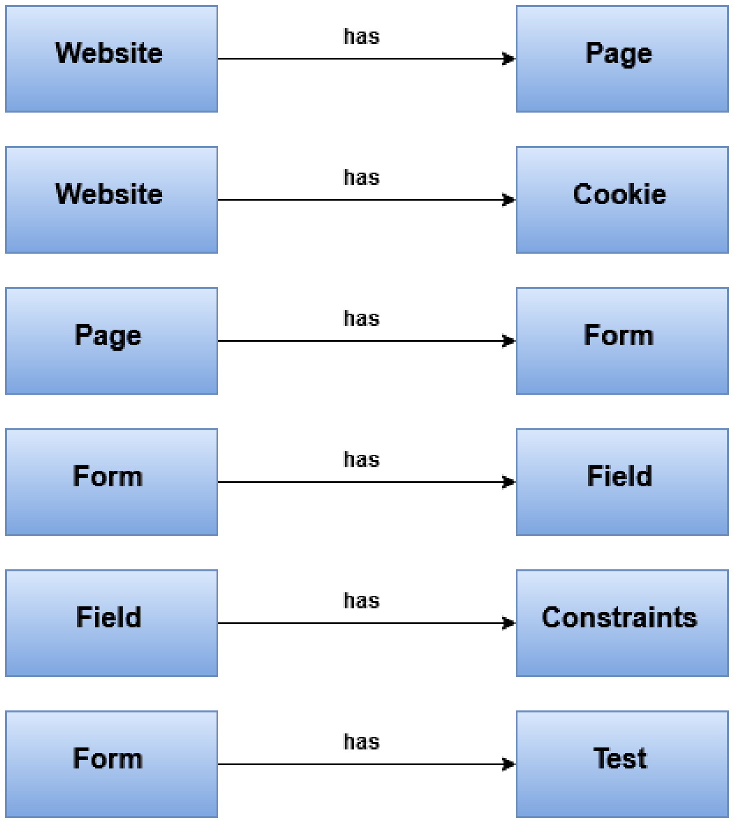


Figure 20: Relationship among data objects

Considering the above relations the following entity relationship (ER) diagram has been constructed.

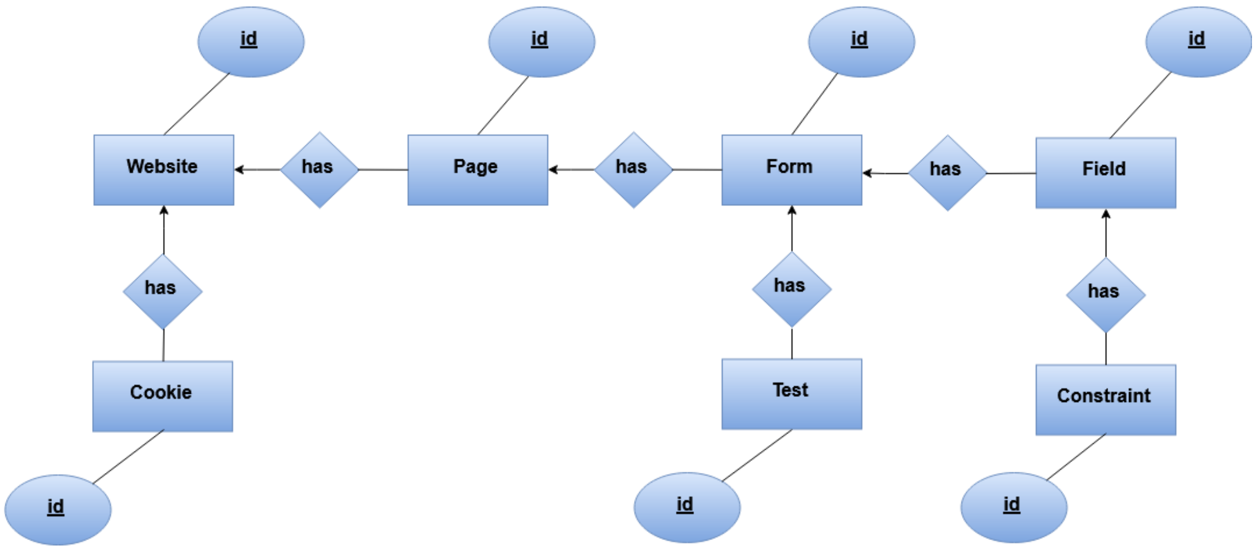


Figure 21: ER diagram

Schema tables

Website

Attributes	Type	Size
id	int	
baseurl	varchar	1000
title	varchar	200

Cookie

Attributes	Type	Size
id	int	
website_id	int	

name	varchar	1000
value	varchar	1000

Page

Attributes	Type	Size
<u>Id</u>	int	
website_id	Int	
url	varchar	2000
screenshot_path	varchar	50

Form

Attributes	Type	Size
<u>id</u>	int	
page_id	Int	
method	varchar	10
form_action	varchar	1000

Field

Attributes	Type	Size
<u>id</u>	int	
form_id	int	
type	varchar	20
name	varchar	100
default_value	varchar	200

Constraint

Attributes	Type	Size
<u>id</u>	int	
field_id	int	
type	varchar	50
value	varchar	50

Test

Attributes	Type	Size
<u>id</u>	int	
form_id	int	
type	varchar	10
input_json	varchar	2000
html_output_path	varchar	50
screenshot_path	varchar	50
time_of_test	datetime	
duration	int	
result	varchar	100

6. Class Based Model

Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined [4].

6.1. Class Identification

After potential analysis of the system the following classes were identified.

1. BaseModel
2. Website
3. Cookie
4. Page
5. Form
6. Field
7. Constraint
8. Test
9. Crawler
10. FormParser
11. Screenshot
12. SQLi
13. JSi

In the above, classes 1-6 have been derived from the data model. The system will be built in a MVC architecture. Thus, the database tables have been mimicked to class models as well for database access and operations. The rest of the classes have been identified through thorough analysis.

The attributes and methods of the classes are given below:

Class	Attributes	Methods
BaseModel		save_to_db(), update(), delete(), as_dict()
Website	id, baseurl, title	
Page	id, website_id, screenshot_path	
Form	id, page_id, method, form_action	
Field	id, form_id, type, name, default_value	
Constraint	id, field_id, type, value	
Test	id, type, form_id, input_json, html_output_path, image_path, time_of_test, duration, result	
Crawler	url, home_url, scanned_urls, not_scanned_urls, running	setup(), crawl(), get_forms()
FormParser	form, baseurl	parse()
Screenshot		snap()
SQLi	form, payload	attack(), analyze()
JSi	form, payload	attack(), analyze()

6.2. Class Cards

Crawler

Responsibility	Collaborator
Finding new URLs	Website, Page
Finding forms	Form

SQLi

Responsibility	Collaborator
Preparing form	AutoFormFill, Form
Injecting fault to target	FileManager, Test, Screenshot

JSi

Responsibility	Collaborator
Preparing form	AutoFormFill, Form
Injecting fault to target	FileManager, Test, Screenshot

AutoFormFill

Responsibility	Collaborator
Automatically fill form	Form, Field, Constraint

BaseModel, Classes Website, Page, Form, Field, Constraint and Test- their only responsibility is to store data. They themselves don’t collaborate with others, but other classes collaborate with them.

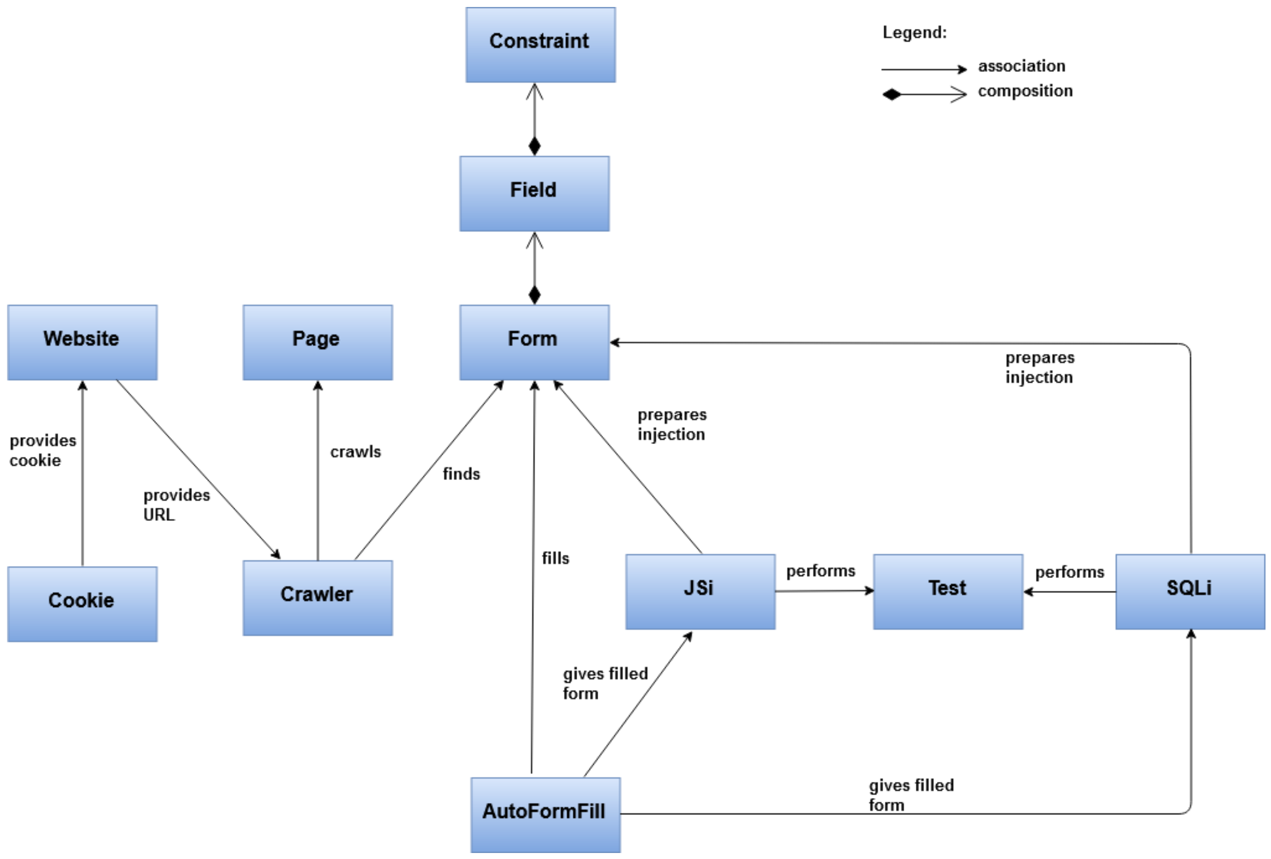


Figure 22: Class diagram

7. Architectural Design

The software architecture of a program or computing system is the structure or structures of the system which comprise-

- The software components
- The externally visible properties of those components
- The relationships among the components [4]

7.1. System in context

The following diagram represents the system in context.

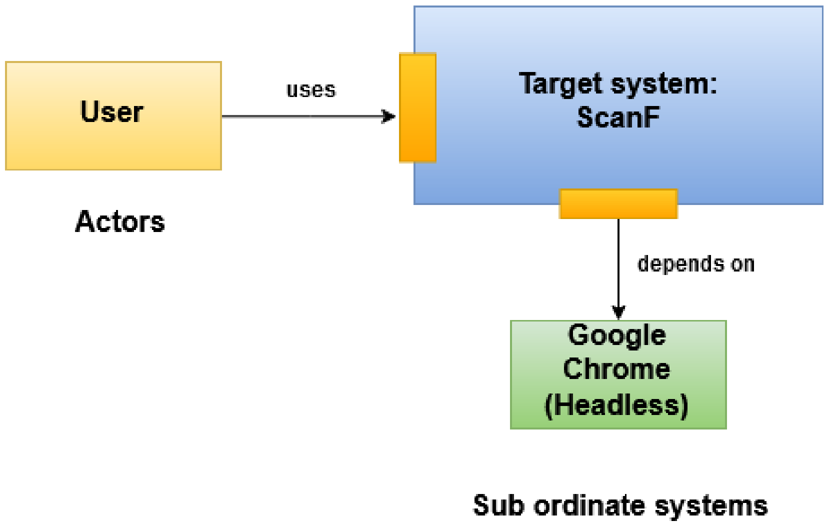


Figure 23: Architectural context diagram

The ScanF system is dependent on headless Chrome. The headless Chrome will be used for testing purposes.

It may be asked why Google chrome has been chosen for testing when these tests can be run in other ways through scripts. The reason is, our system would like a screenshot of the system with the JavaScript scripts executed. Moreover taking screenshots with a browser rather than manually setting up HTML and CSS is easier. Also the ScanF tool is interested in a total browser behavior, so, using a browser helps.

7.2. High level component level design

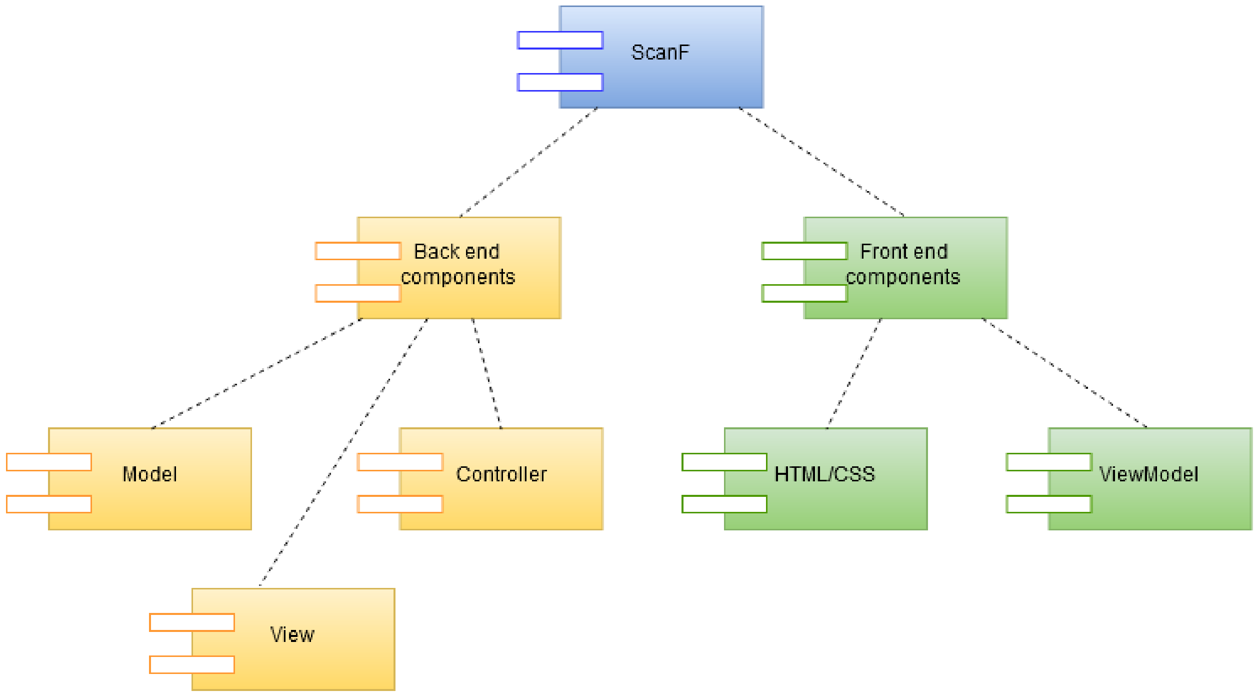


Figure 24: High level component level design

The entire system will have some backend components and some front end components. The backend be built around an MVC architecture. The model and view classes have been already identified in the previous sections. The controller classes have only one responsibility- to receive requests from the UI and forward them to another class for performing an operation. The front end will have some HTML, CSS and ViewModel components. The ViewModels will be made with the Vue framework.

8. Interface design

The UI of the ScanF tool is expected to be simple but very powerful. The UI will be a single page application. In fact, there will be only one page from where the user can observe and monitor everything. Contents of inner divs will change dynamically according to user actions.

The wireframe of the expected UI is something like this.

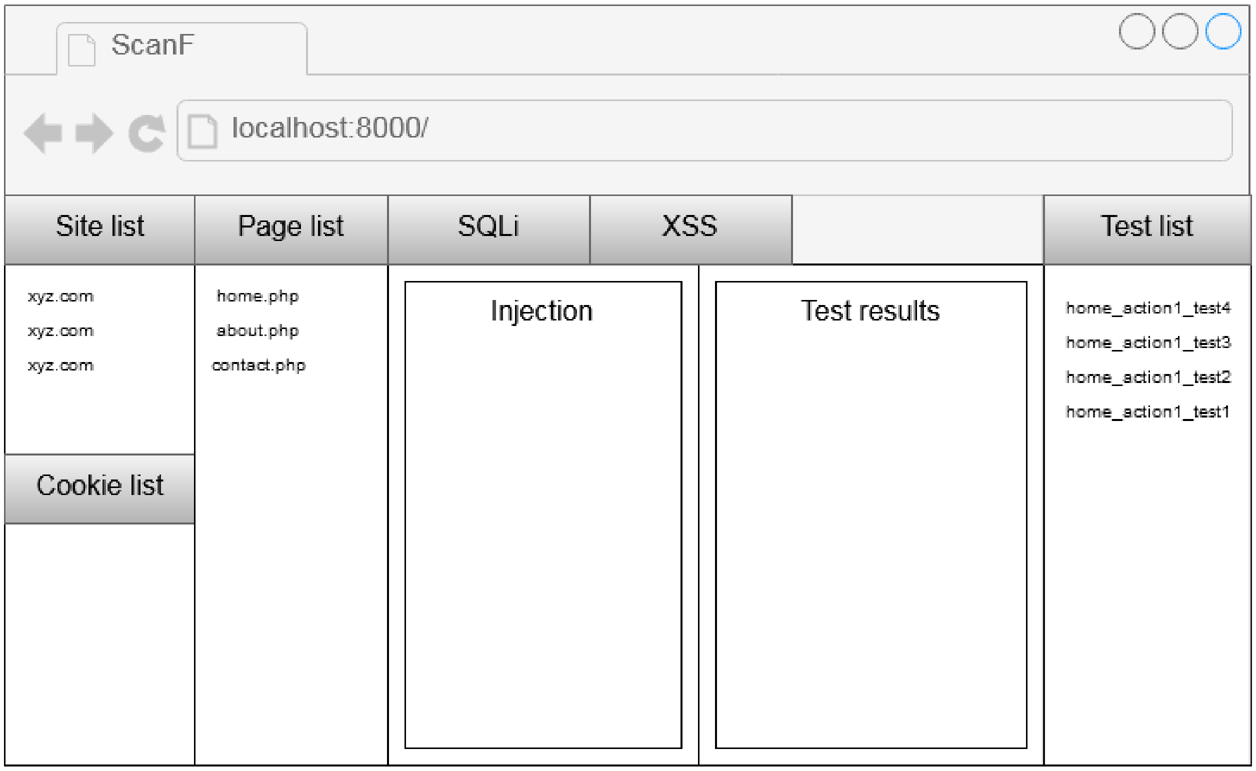


Figure 25: Interface design

To run the program, a user will run a script which will open a local HTTP server. Visiting through a browser will show this interface.

9. Implementation

The project is implemented as a web application. There is a web based UI, and a backend that does the core functionalities. The implementation of the two components are described below.

9.1. Front end implementation

The main function of the front end is to interact with the user and show data by fetching it from the backend. To do these, an MVVM (Model View ViewModel) architecture has been followed in the front end. To achieve an MVVM architecture, apart from the usual HTML, CSS and JavaScript, the following libraries were used-

1. Axios (Model)
2. Bootstrap3, hacker-bootstrap (View)
3. Vue.js 2 (ViewModel)

Axios is an HTTP request library that fetches data from a server. In our case, axios gets data from a REST API and constructs a JSON object which acts as our Model.

The view in our application is done through bootstrap3. Bootstrap3 makes the user interface responsive to devices of different height/width ratios. Although our interface is not intended for mobile devices, the responsiveness will be useful for large device of different ratios.

Apart from bootstrap we have used the hacker-bootstrap template as an extra layer over bootstrap. This gives the makes the interface look professional looking and feel geeky.

The core of the front end is done through vue.js. We have used mainly the following features of vue.js-

- Model binding ability: With model binding, two-way data bindings on form input, textarea, and select elements can be created. This eases receiving data from the user
- Automatic DOM modifications: A change in model is automatically reflected in the DOM. Through this, more time can be spent thinking about app logic and less on UI rendering.
- Separation of concerns: Through this, the application has been split over several components. This makes development and management easier.

The front end code can be found at https://github.com/rafed123/ScanF/tree/master/client

The front end is structured as follows-

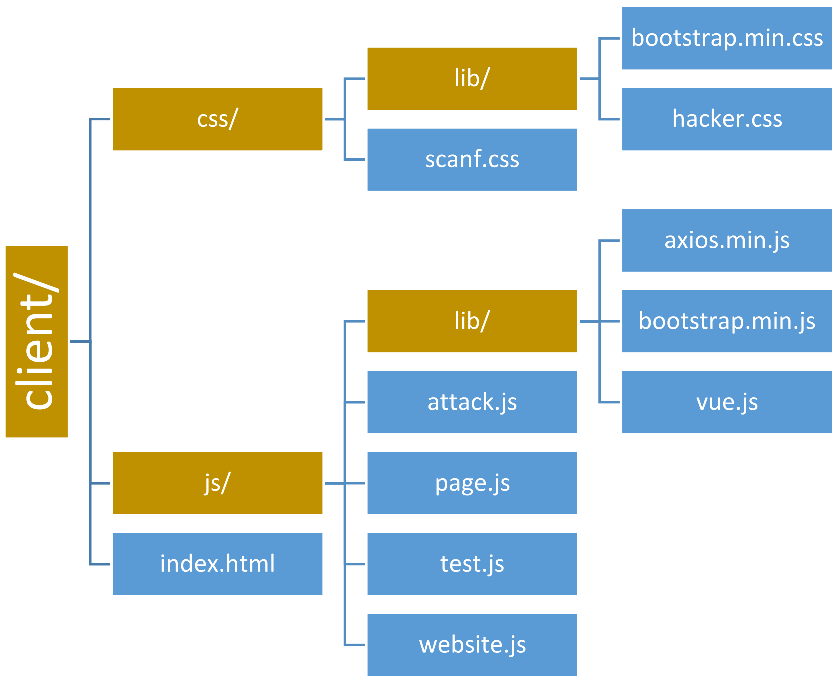


Figure 26: Front end structure

Everything inside a lib directory is a library. In the js directory attack.js, page.js, test,js and website.js are vue components. Scanf.css contains custom CSS for the UI. Index.html contains the HTML template of the entire front end.

9.2. Back end implementation

The main activity of the backend is receive data from the client, process it and send it back to the client. The backend has been built as a REST API. To build the API and MVC architecture has been followed. To build the backend the following libraries/technologies have been used-

1. beautifulsoup4 (v4.6): This is a library for parsing HTML content. This library has been used to parse forms and its fields from an HTML file so that attacks can be performed upon them.
2. Flask (v1.0): Flask is a micro web framework for building python based web servers. This is the main building point of the backend. Then entire backend is built around Flask.

3. Flask-Restful (v0.3.6): This is an extension to the Flask framework. With Flask-Restful, making REST based services become a lot easier. With Flask-Restful, controller and view part of the MVC architecture is achieved. The view in our application is plain JSON.
4. requests (v 2.20): This library is used to make HTTP requests. With this library sites are crawled for HTML content.
5. selenium (v3.141): This is a web driver library for running browsers. This library has been used to drive Google chrome and perform attacks on websites.
6. Sqlite (v3): Sqlite is a lightweight database. All information of the ScanF system is stored in a sqlite database instance.
7. SQLAlchemy (v1.2): This is a library to make database calls easier. SQLAlchemy provides an easy way to make ORMs (Object Relational Mappers) which eases CRUD operations on a database. With SQLAlchemy, the model part of the MVC architecture is achieved.
8. urllib3 (v1.24): This is a library similar to requests, but it has some additional features. This has been mainly used to parse parts of URLs.

The backend of the project can be found at https://github.com/rafed123/ScanF. The backend is structured as the following

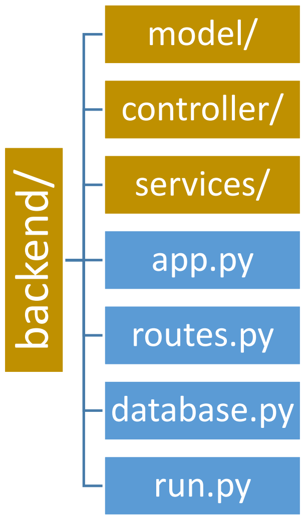


Figure 27: Backend structure

Each directory/file in the structure is explained below-

1. Model: This directory contains all the ORM models. The following models are present-
    a. Basemodel.py (All models inherit this class so they can perform CRUD operations on a sqlite database)
    b. Website.py
    c. Cookie.py
    d. Page.py
    e. Form.py
    f. Field.py
    g. Contraint.py
    h. Test.py
    i. Sql.py
2. Controller: Controllers define what kind of HTTP verbs a REST endpoint will have and what operations will they perform. The following classes reside in the controller directory-
    a. WebsiteController.py
    b. CookieController.py
    c. PageController.py
    d. FormController.py
    e. ScreenshotController.py
    f. SqliController.py:
    g. XSSController.py:
    h. TestController.py
3. Services: This directory contains the core operations related to the main purpose of the application. The following classes reside in this directory-
    a. Crawler.py
    b. Form_parser.py
    c. Auto_Form_Fill.py
    d. Screenshot.py
    e. Sqli.py
    f. Xss.py

4. Routes.py: This file defines all the endpoints of the application and which class will be invoked upon receiving request on a particular endpoint.
5. Database.py: This file creates a database for the system if a database has not been created for the system yet.
6. Run.py: This file initializes everything needed to run the tool and then starts the backend server for the web client to use.

Thorough testing has been done on the backend. Discussion about testing has been done in the next section.

10. Testing

This section discusses the testing approach for this tool and what tests has been done to ensure quality.

The ScanF tool's core functionalities are to perform SQLi and XSS attacks. Apart from this two functionalities there are other modules that builds up a huge ecosystem that is aimed at making the software well designed and easy to use. Both the core functions and the additional functions for satisfying good design were tested for correctness and performance.

10.1. Approach to Testing

To test the application unit test and blackbox testing has been done. Unit test has been done on the core features, i.e. the services module of the backend. Blackbox testing (equivalence partitioning) has been done on the backend to test database integration and correctness of crud operations.

10.2. Test tools

For testing, the following tools have been used:

1. Pytest (for unit testing)
2. cURL (for testing API calls)
3. Travis CI (for continuous integration)

10.3. Test items

The following items have been tested:

1. API calls
2. CRUD operations
3. Services-
    a. Website crawler
    b. Auto form fillup
    c. Response analyzer

10.4. Test cases

Test case: 1

Test case name: Controller test

Description: Try to perform CRUD operations on database through the REST API (Equivalent partitioning blackbox test)

Steps	**Action**	**Input test data**	**Expected response**	**Pass/Fail**
1.	Insert website to db	url	Inserted	Pass
2.	Retrieve websites	None	Retrieved	Pass
3.	Delete website	Website_id	Deleted	Pass
4.	Insert page to db	Page_url, website_id	Inserted	Pass
5.	Retrieve pages	Website_id	Pages of website with that id	Pass
6.	Delete page	Page_id	Deleted	Pass
7.	Insert cookie	Name, value	Inserted	Pass
8.	Update cookie	Cookie id, name, value	Updated	Pass
9.	Delete cookie	Cookie id	Deleted	Pass
10.	Insert form	Page_id, action, method	Inserted	Pass
11.	Retrieve form	Page_id	Forms of a page with that page id	Pass
12.	Delete form	Form_id	Form deleted along with cascading fields and constraints	pass
13.	Insert fields	Form_id, form json	Inserted	Pass

14.	Insert constraints	Field id, constraint json	Inserted	Pass
15.	Insert tests	Form_id, test result	Inserted	Pass
16.	Retrieve tests	Form_id	All tests of a form with that id	Pass
17.	Delete tests	Test id	Deleted	Pass

Test case: 2

Test case name: Crawler test

Description: Test whether the crawler can parse a website correctly (Unit test)

Steps	**Action**	**Input test data**	**Expected response**	**Pass/Fail**
1.	Get website internal links	Website url	Correct number of links	Pass
2.	Get forms	Page link	Correct number of forms, along with its fields and constraints	Pass

Test case: 3

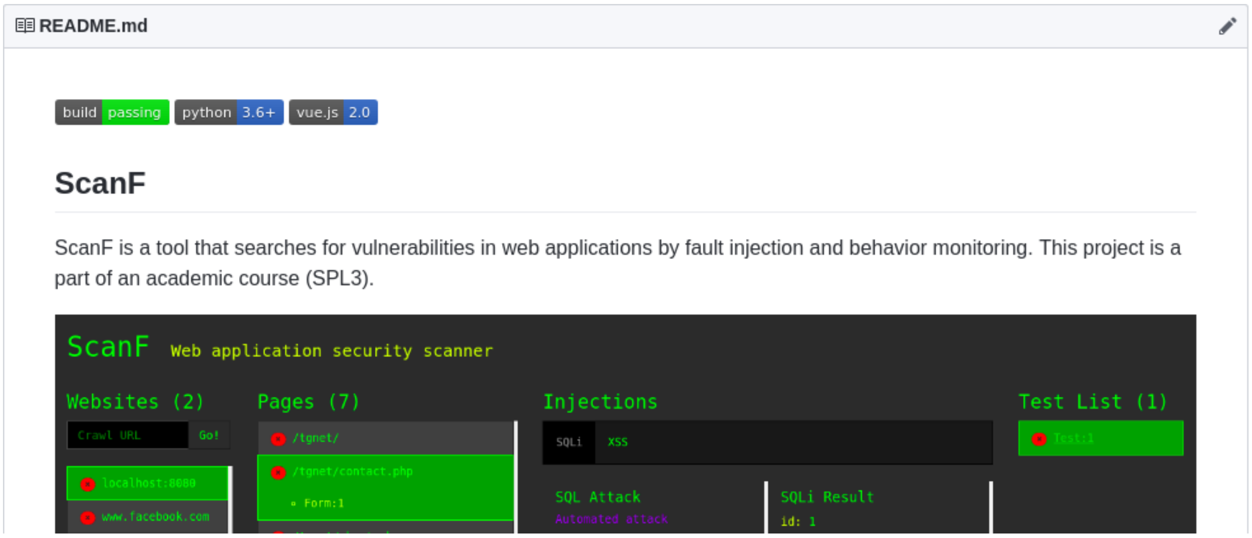
Test case name: Response analyzer

Description: Test whether the response after delivering a payload has message indicating vulnerability (Unit test)

Steps	**Action**	**Input test data**	**Expected response**	**Pass/Fail**
1.	Test a form which shows error messages in payload submission	Form id, payload	Vulnerable	Pass

2.	Test a form which does not show error messages in payload submission	Page link	Not vulnerable	Pass

The tests of the tool can be found in https://github.com/rafed123/ScanF/tree/master/tests. The repository is linked with Travis CI. So, every time a push is sent, the tests are automatically run. This ensures integrity in every push. Following is a screenshot of the repository where the test pass/fail status of the repository is show (Build: passing).


README.md

build passing | python 3.6+ | vue.js 2.0

ScanF

ScanF is a tool that searches for vulnerabilities in web applications by fault injection and behavior monitoring. This project is a part of an academic course (SPL3).

11. User Manual

ScanF is a tool to run SQL injections and cross site scripting attacks. These attacks can be sometimes tricky and complex to perform. ScanF makes it easy to run and understand these attacks. To run the tool, follow the instructions below.

11.1. System requirements

The following system requirements must be met before installing the tool.

- Linux OS (Debian based distributions)
- Minimum 2 Gb RAM
- Minimum 1 Gb of remaining disk space
- [Optional] Non proxy/non VPN internet connection (for best performance)

11.2. Install program dependencies

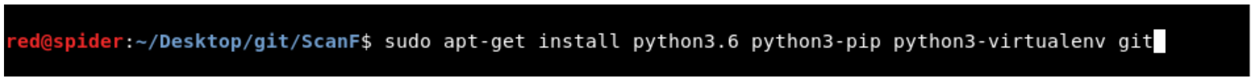
Follow the steps below to meet the requirements needed by the tool.

1. Install system requirements

   Install Google chrome following this tutorial https://linuxize.com/post/how-to-install-google-chrome-web-browser-on-ubuntu-18-04/

   Install Python3.6, Python3-pip, Python3-virtualenv and Git

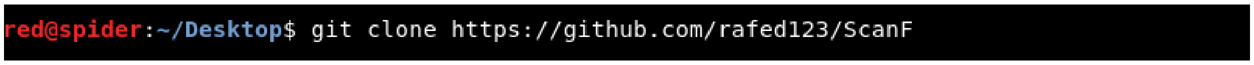
   sudo apt-get install python3.6 python3-pip python3-virtualenv git

```
red@spider:~/Desktop/git/ScanF$ sudo apt-get install python3.6 python3-pip python3-virtualenv git
```

2. Open terminal and clone the repository

   git clone https://github.com/rafed123/ScanF

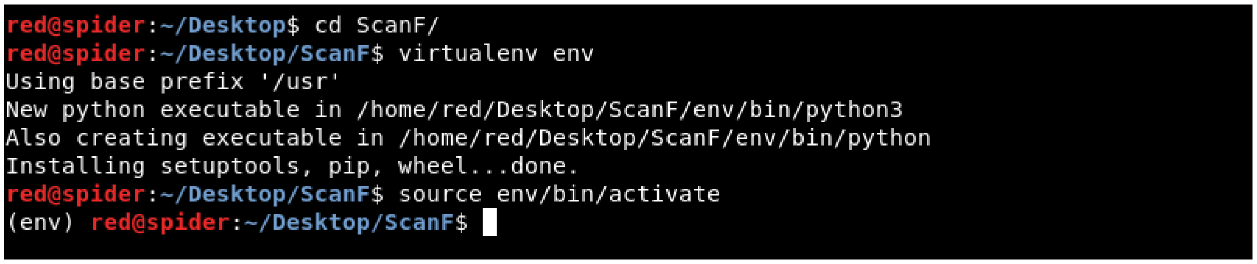
```
red@spider:~/Desktop$ git clone https://github.com/rafed123/ScanF
```

3. [Optional] Make a virtual environment. Virtual environment provides isolated environment for python to run. This is not mandatory but highly recommended. Navigate to the repository and run-

   cd ScanF

virtualenv env

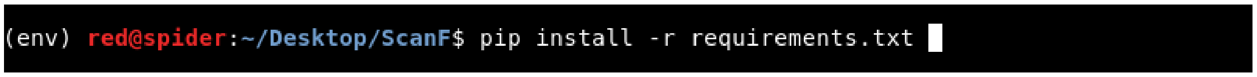
source env/bin/activate



```
red@spider:~/Desktop$ cd ScanF/
red@spider:~/Desktop/ScanF$ virtualenv env
Using base prefix '/usr'
New python executable in /home/red/Desktop/ScanF/env/bin/python3
Also creating executable in /home/red/Desktop/ScanF/env/bin/python
Installing setuptools, pip, wheel...done.
red@spider:~/Desktop/ScanF$ source env/bin/activate
(env) red@spider:~/Desktop/ScanF$
```

4. Install dependencies

   pip install –r requirements.txt

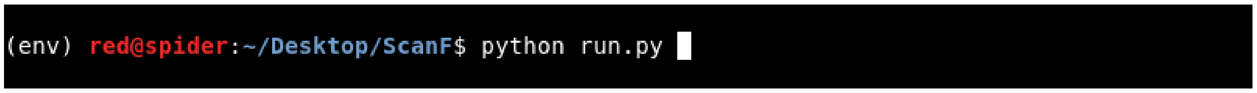


```
(env) red@spider:~/Desktop/ScanF$ pip install -r requirements.txt
```

11.3. Using the tool

1. Run the application-

   python run.py

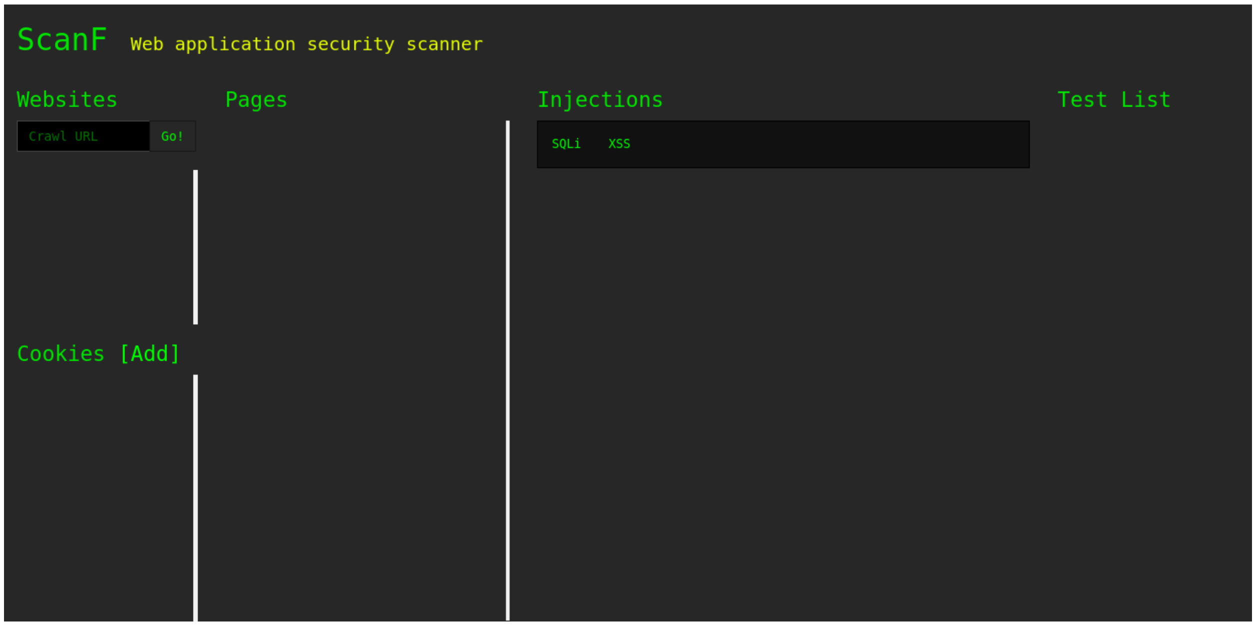


```
(env) red@spider:~/Desktop/ScanF$ python run.py
```

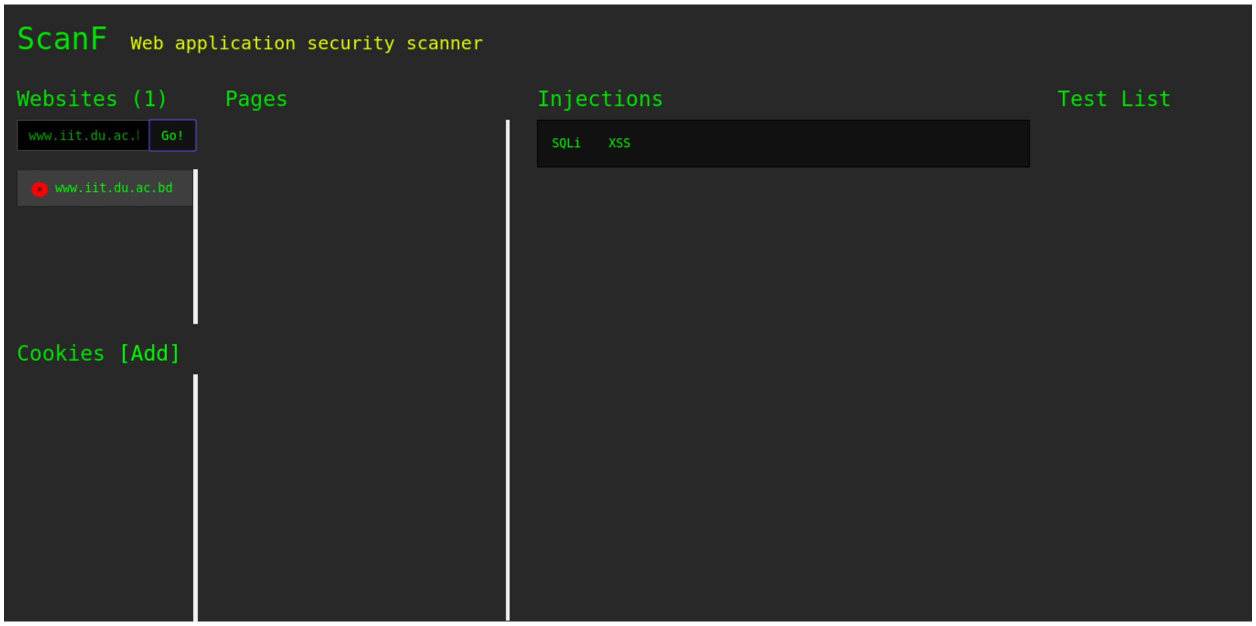
2. Open the application in a browser (preferably Google chrome) and go to

   http://localhost:5000

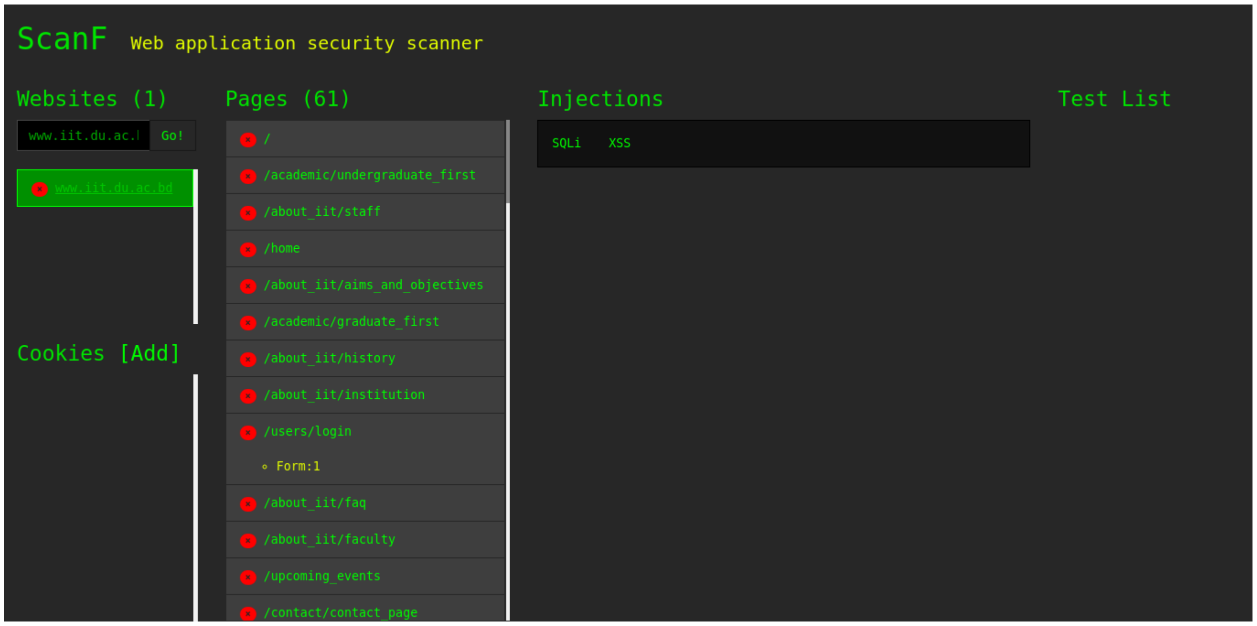
   A user interface like the following will appear. On the top left bar enter the URL of a site you want to scan and click Go.



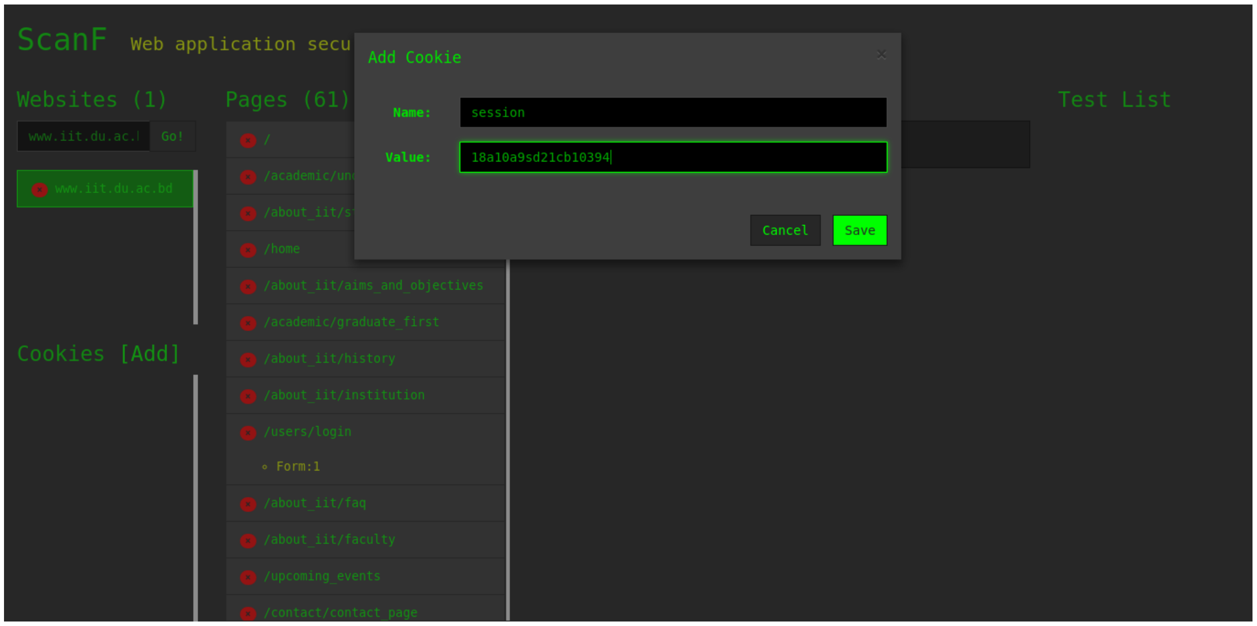
3. Upon clicking Go, ScanF will start crawling the site and find all the links it can. If a scan takes a long time (which will happen in most cases), stop the scan by clicking on the cross where the Go button was before. Now you can see your scanned site on the website list.



4. Click on a website item to view all its crawled pages.



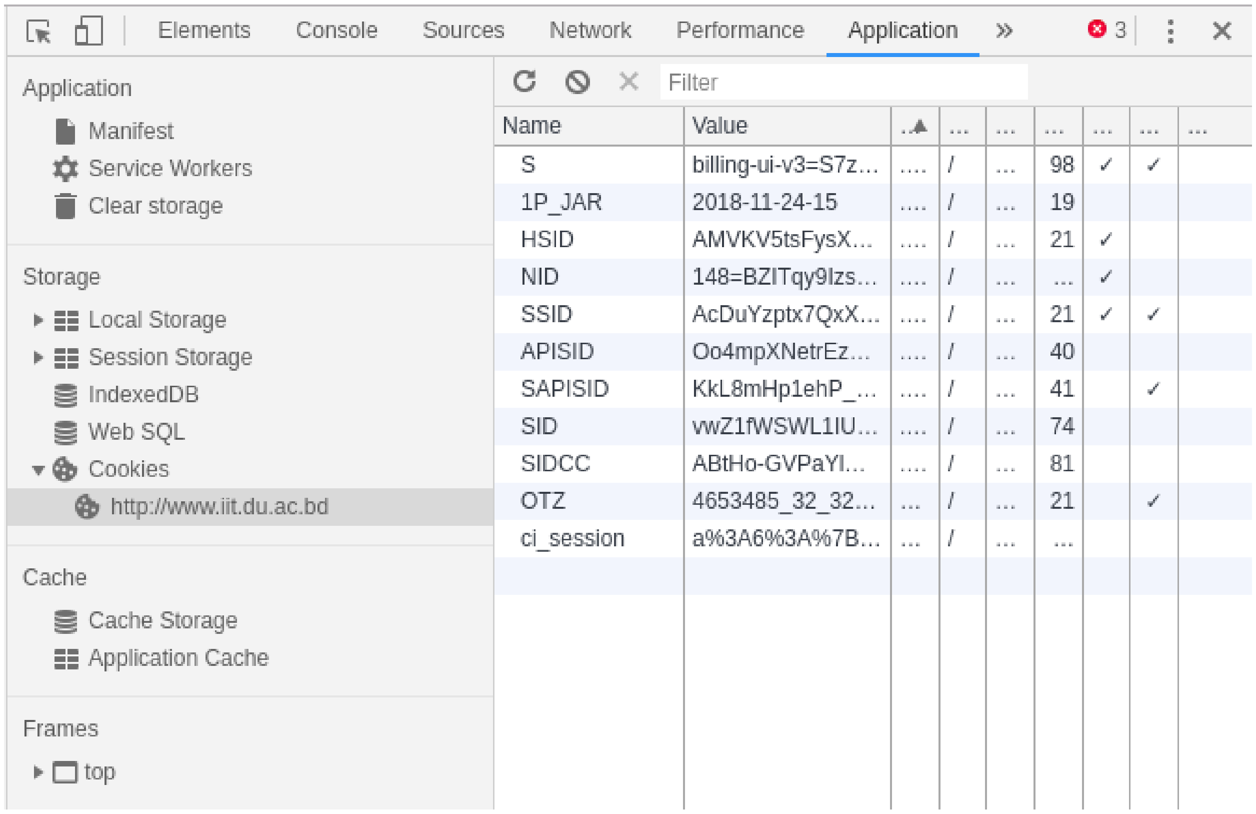
5. Sometimes you may need to crawl login protected links. To do this, you can add cookie by copying it from the browser after logging in yourself. Click on [Add] on the Cookie panel and the following form will appear.



Name: session

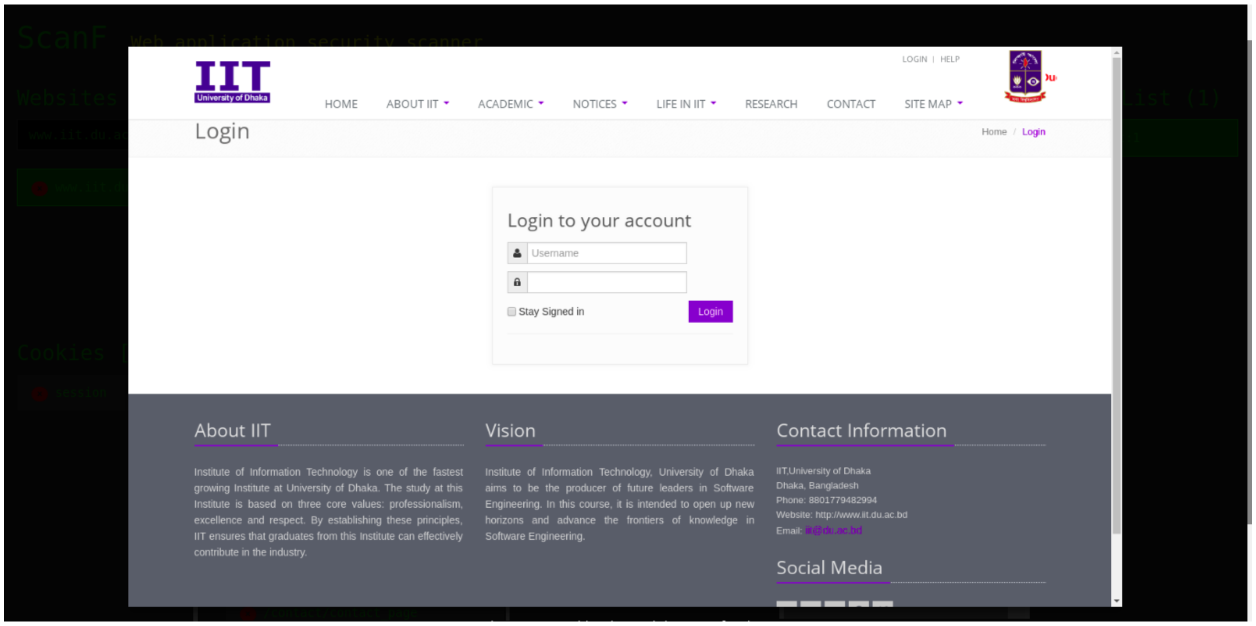
Value: 18a10a9sd21cb10394

To get a cookie go to a site (through Google chrome) and press F12. Go to "Application" tab and go to "Cookies". Here you can find name/value pair of a cookie. You can copy cookies from here and paste it on the form.



Name	Value	...	...	...	...	...	...	...
S	billing-ui-v3=S7z...	....	/	...	98	✓	✓	
1P_JAR	2018-11-24-15	....	/	...	19			
HSID	AMVKV5tsFysX...	....	/	...	21	✓		
NID	148=BZITqy9Izs...	....	/	...	...	✓		
SSID	AcDuYzptx7QxX...	....	/	...	21	✓	✓	
APISID	Oo4mpXNetrEz...	....	/	...	40			
SAPISID	KkL8mHp1ehP_...	....	/	...	41		✓	
SID	vwZ1fWSWL1IU...	....	/	...	74			
SIDCC	ABtHo-GVPaYI...	....	/	...	81			
OTZ	4653485_32_32...	...	/	...	21		✓	
ci_session	a%3A6%3A%7B...	...	/	...	...			

6. It is more worthwhile to see the site itself rather than the its link. You can do this easily from within ScanF. Simply click a page link and wait while the site loads.



Login

Login to your account

Stay Signed in

About IIT

Institute of Information Technology is one of the fastest growing Institute at University of Dhaka. The study at this Institute is based on three core values: professionalism, excellence and respect. By establishing these principles, IIT ensures that graduates from this Institute can effectively contribute in the industry.

Vision

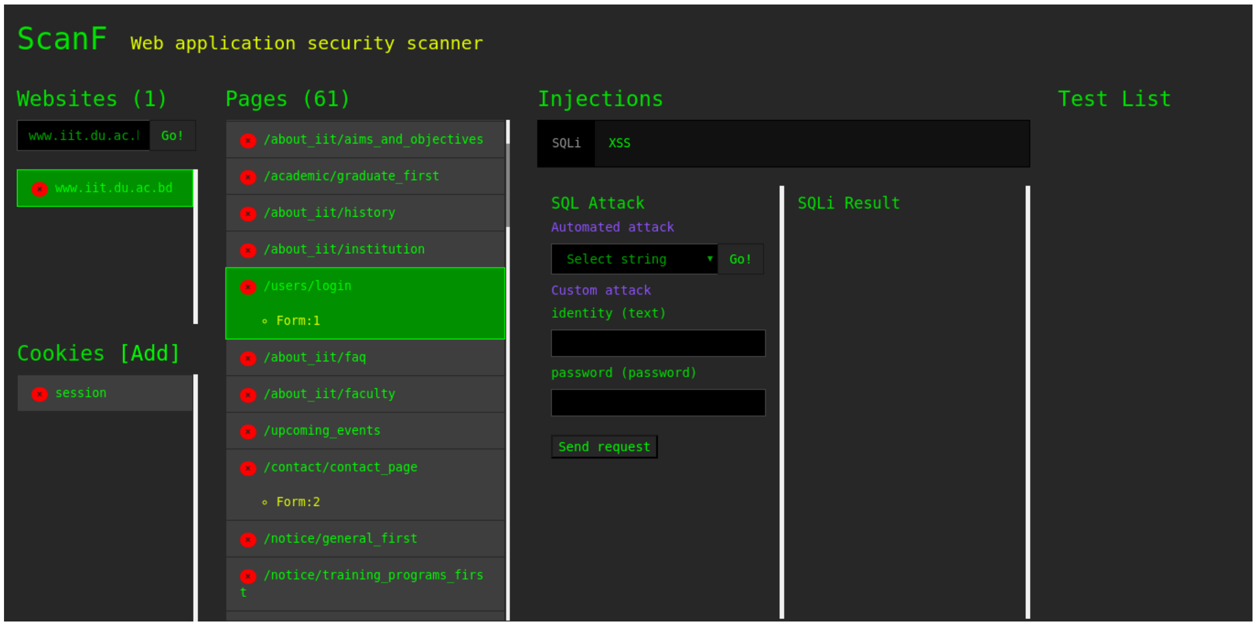
Institute of Information Technology, University of Dhaka aims to be the producer of future leaders in Software Engineering. In this course, it is intended to open up new horizons and advance the frontiers of knowledge in Software Engineering.

Contact Information

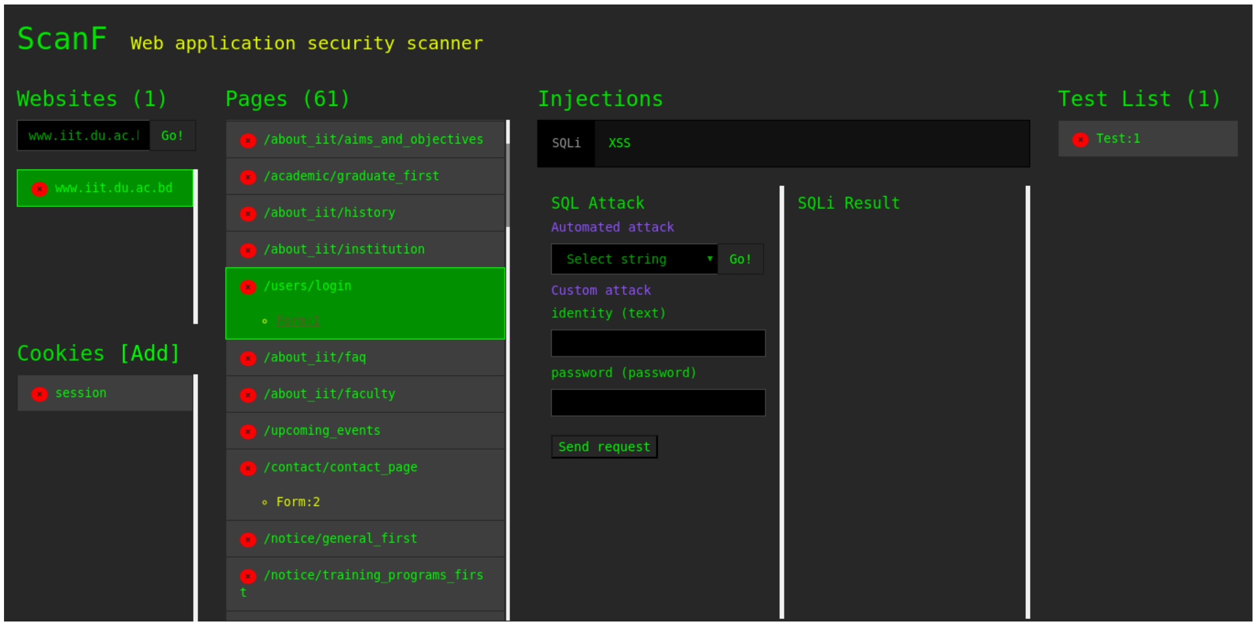
IIT,University of Dhaka
Dhaka, Bangladesh
Phone: 8801779482994
Website: http://www.iit.du.ac.bd
Email: iit@du.ac.bd

Social Media

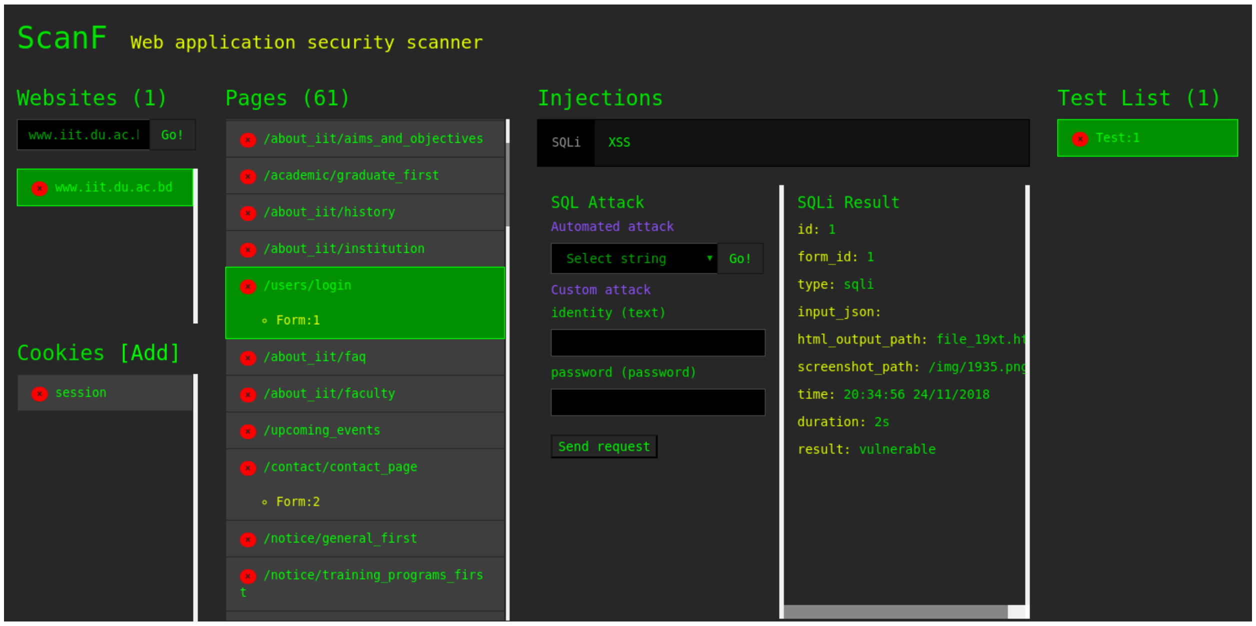
7. On the list of pages you can see which pages contain forms. To test a form click on it and a view like the following will appear.



8. On the Injections panel, you can choose which kind of attack you want to perform. For SQL injections there are two modes- automated and manual injection. In automated mode, choose a SQL string and click Go to perform automated attacks. Or you can manually insert payloads yourself in the custom mode and send requests to test. Upon completion of tests, the test lists will show some rows.

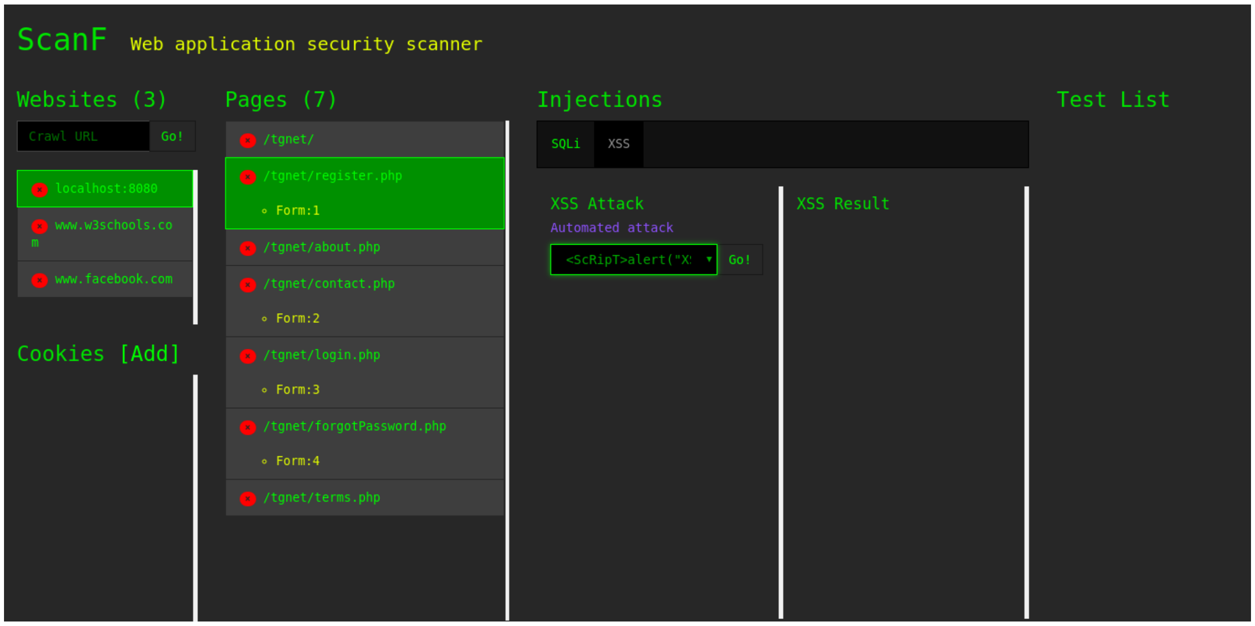


9. Click on the test item to see the results of the test.



```
id: 1
form_id: 1
type: sqli
input_json:
html_output_path: file_19xt.ht
screenshot_path: /img/1935.png
time: 20:34:56 24/11/2018
duration: 2s
result: vulnerable
```

10. The XSS tab also works the same way as SQL injection. Select a XSS payload from the list and click on go. Results will appear on the XSS Result tab. Happy scanning!



11.4. Troubleshooting guide

- **Error alerts**: If the browser continuously alerts “An error occurred”, check if the local server has stopped running. If it has stopped, restart it.
- **Error alerts but local server running:** Check if your internet connection is up or not. Also, if this case is for an attack check if the target website is behind a firewall.
- **Broken UI:** If the UI seems out of place, try setting the browser zoom level to 100%. The UI is not supported yet for small devices or with high zoom level.

12. Conclusion

I am pleased to say that I have had a wonderful experience developing this tool. Expressing practical work on paper is a difficult task. Despite this I have tried my best to describe my work accurately in this paper.

This paper covers all the aspect of building this tool from the very beginning to the end. At first the software requirement specifications are discussed. Based on these requirements, the design plan made is laid out. Then, according to the design, an implementation was made. The source code of the implemented project can be found at https://github.com/rafed123/ScanF. The implemented tool was finally tested. The test plan and results are also presented in this document. And finally, so that anyone can use this tool properly, a user manual has been provided.

I have learnt many things during the development of this tool. Mostly I have learnt about new security concepts. Apart from that, I have also learnt some modern techniques that assist building applications for the web.

This tool can be further extended in many ways. Currently it has support for only a few SQLk and XSS payloads. More can be added. Also new types of attack can be added such as CSRF, LFI, etc.

I end this paper with the hope that this tool will be of use to anybody who wants to assess security in their website.

References

[1] Brobin, Hacker css, https://brobin.github.io/hacker-bootstrap/, last accessed on 20th November, 2018

[2] Vue.js, https://012.vuejs.org/guide/, last accessed on 20th November, 2018

[3] SQL-Injection-Payloads, https://github.com/trietptm/SQL-Injection-Payloads/, last accessed on 10th November, 2018

[4] Pressman, Software Engineering: A Practitioner's Approach