Model Based Security Testing for Web Applications

Submitted in partial fulfillment of the requirements of the degree of

Master of Technology (M.Tech)

by

Anbarasan K

***Roll no.* 163050071**

Supervisor:

Prof. R.K. Shyamasundar

Department of Computer Science & Engineering

Indian Institute of Technology Bombay

2018

Dissertation Approval

This project report entitled **"Model Based Security Testing for Web Applications"**, submitted by **Anbarasan K** (Roll No. **163050071**), is approved for the award of degree of **Master of Technology (M.Tech)** in Computer Science & Engineering.

RK Shyamasundar

Prof. R.K. Shyamasundar
Dept. of CSE, IIT Bombay
Supervisor

G. Sivakumar

Prof. G. Sivakumar
Dept. of CSE, IIT Bombay
External and Internal Examiner

[illegible]

Prof. Bernard L. Menezes
Dept. of CSE, IIT Bombay
Internal Examiner

Date: 29.. June 2018

Place: JUNE 29, IIT BOMBAY

Declaration of Authorship

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature: Anbarasan ....

Anbarasan K

163050071

Date: 29 June 2018

Abstract

Testing of web applications would have become manageable if we had a functional specification of web applications. Due to the underlying issues with browsers and applications being patched regularly it is almost impossible to set realistic functional specifications and system specifications. Thus, vulnerability scanners to test web security will result in false positives, or false negatives. The above disadvantages can be circumvented by model-based testing approach. In this dissertation, we propose a method for web application security using model-based testing approach. The proposed method has been implemented and tested. Model-based testing approach is adapted as it is easy to automate the testing process, and adaptive to changes. Some of the modelling tools like WebSphinx, and PHP-Crawler generate a model based on the static elements in web page. Modern web applications are dynamic in nature due to AJAX technology. Our approach, uses crawljax, a open source modelling tool, and generates a model based on dynamic event handler, then generates test cases for security based on the model of the web application, and execute them in a browser to validate the model for security vulnerabilities. Our case study of DVWA-Damn Vulnerable Web Application that contains security vulnerabilities has shown advantages of our tool approach. We have performed experiments with our test framework in a sample space of 15 web applications. We opine that our testing tool is capable of capturing security vulnerabilities like XSS and SQL Injection in the tested web applications in an automated manner.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Web Applications

Web Applications are application programs that are stored in remote servers and delivered over the Internet using browser interfaces. The user interface of web applications are built using the interplay of HTML, CSS and JavaScript. Javascript being a highly dyanamic language, web applications are dynamic with interactive elements and some applications require server side processing. Web Applications require a server to handle client requests and a database to store the information. Web Applications include e-commerce websites, online forms, content management systems and email programs[1].

1.2 Security Testing

With the increasing complexity of web systems, and more content moving from offline to online platform, security testing has become a critical activity of web application development life cycle. The focus of security testing of web applications is to maintain the confidentiality of data, prevention against information leakages. This is performed by checking web application behaviour on injecting malicious data. Security Testings aids in exposing security vulnerabilities like XSS, SQL Injection, file inclusion, URL injection[3].

1.3 Web Application Modelling

A model is an abstraction or simplification of the behavior of the application under test(SUT). The model is captured in a machine with the purpose of acting as a test sequence (trace) generator. The model represents the architecture of the web application. A web application can be modelled using finite State machine or labelled transition system approaches[7].

1.4 Model Based Testing

Model-based testing is a technique for designing and executing applications to perform testing (includes Test cases to be executed on every object in the model). In Model Based Testing, test cases are generated automatically and systematically from the model of the system under test. There are three step in Model based Testing. Firstly, a model of the SUT is built from informal requirements or from the functionality of the SUT. Secondly, execution traces of the model are used to generate test cases. Since there can be infinitely many/long execution traces be present, test selection criteria is applied reduce the testcases. Lastly, using the test model and the test selection criteria, Test cases are derived for model based testing[7].

1.5 Model Based Security Testing

Model based testing can be adapted to test security of web applications. Model based testing for functionality usually generates positive test cases for validation of web applications. To perform security testing, negative test cases and attack scenarios have to be derived. Attack scenarios include generation of malicious code or data to retrieve unauthorized data from the web server. This involves two steps- Modeling activity to capture the behavioral aspects and/or architecture of the web application. The next step is to define the test purposes- in this case to test the security in particular the type of attack on web security.

Chapter 2

PROBLEM STATEMENT

The growth of Internet and evolution of web application technologies has led to the vulnerabilities in security- data confidentiality, data integrity and service availability. Manual testing of security in web applications is time consuming and its difficult to perform in case of regression testing. Web Vulnerability scanners can be used to detect vulnerabilities but they often led to generation of false positive and false negative results. A desirable solution to perform security testing is automated security testing. Automated security testing has its challenges in terms of deriving proper functional behavior of the web applications. This is due to the lack of proper documentation of web applications or various technologies used for developing web applications. Modern web applications are no more static but dynamic due to the extensive use of javascript and CSS. With javascript, functionalities can be added in the client side of the web applications which makes calls to the server using (Asynchronous javascript) AJAX. These calls needs to be validated and tested thoroughly to prevent information leaks.

To overcome the above difficulties in security testing of application, there is a need of method for automated testing of vulnerabilities of software that can be realized testing it with a model can overcome the above challenges. This approach is called model based security testing. We derive a model for web applications handling AJAX(Javascript and XML) calls and test security vulnerabilities based on the testcases derived from model.

The thesis is organized as follows:
Chapter 3- Literature Survey, Chapter 4- Web Application Security Vulnerabilities, Chapter5- Our approach of modelling web applications for security testing, Chapter 6- Modelling of Web Applications via State Machine Chapter 7- Security Test Framework, Chapter 8- Experimentation and Results, Chapter 9- Related Work, Chapter 10- Conclusion and Chapter 11- Future Work

Existing approaches like SQLMap performs security testing without out a model. Our approach being model driven provides exhaustive coverage of testing web applications.

Chapter 3

LITERATURE SURVEY

In this chapter, we give a brief overview of the existing tools for web testing.

3.1 WebMate: A Tool for Testing Web 2.0 Applications

Webmate[1] is a functional testing tool for Web 2.0 applications. Webmate automatically crawls through the Web applications performing regular application testing as well as cross browser compatibility testing. The input to the webmate tool is the url of the website to test and it creates a usage model by crawling and triggering user actions with a javascript event handler. This usage model is extracted and different browsers and the model is compared with a reference model to capture cross-browser compatibility issues. Webmate also checks for HTML and CSS for browser specific issues. Results of the tests are reported back to the tester. Webmate approach as opposed to manual testing generates testcases automatically for functional testing of web applications.

3.2 Search-based security testing of web applications

Biofuzz[3], search-based security testing tool proposes a blackbox testing technique to inject malicious SQL data to detect SQL Injection vulnerabilities. It injects malicious SQL query in form inputs and based on the feedback of SQL query it detects whether the attack is successful. It identifies target input parameters such as form input fields, and generate form inputs to ther server. The tool Biofuzz, is bound to MYSQL database and PHP as server. Biofuzz tool finds all the input fields with a crawler to detect input fields vulnerable to SQL attacks. It uses evolutionary algorithms to mutate the SQL inputs to test all possible combinations of SQL queries.

3.3 Crawling AJAX-Based Web Applications through Dynamic Analysis

Crawling AJAX is more difficult than classical multi-page web applications. In Multi-page web appliations, each page has a unique URL assigned to them but with AJAX pages are dynamic which multiple content can be assigned to same URL. Extraction of hyperlinks and creating a model does not accurately represent the architecture of web applications. Instead of hyperlinks javascript handle can be used to navigate the application and create a model. The web applications are dynamically represented through the Document Object Model(DOM). The User Interface changes are accurately represented by DOM changes instead of URL changes[4,5].

3.4 Issues with existing tools

Webmate is model based testing tool only for functionality testing of web applications. The security testing feature is not offered. MBVT Tool is a model based vulnerability testing tool but it requires functional specification of web applications for its model and realizing a functional specification for web applications is not feasible because of web application being regularly patched.

Chapter 4

WEB APPLICATION SECURITY VULNERABILITIES

In this chapter, we provide an overview of the most prevalent security vulnerabilities(among top 10 vulnerabilities) in web applications.[8]

4.1 XSS-Cross Site Scripting

Cross-Site Scripting or XSS[6] is one of the most prevelant security attacks in the world.Cross Site scripting or XSS attack targets end-users. User input fields such as form field, url parameter, cookie are vulnerable to XSS attacks. The fields are used by the server to produce responses. An attacker can inject malicious data using a script, usually written in JavaScript, which will be executed by an end-user browser through the input fields in web applications. If the input fields are not validated and sanitized for malicious scripts, then the web applications are vulnerable to attack.

4.1.1 Reflected XSS-Cross Site Scripting

In reflected XSS attacks[6], the malicious script is executed immediately and response containing malicious requests is provided to the end-user. This kind of attack is non-persistent and affect the user executing script in fields such as a search box query or the user clicking a link from a malicious URL. Spam emails containing the malicious URL or Javascript can also initiate a Reflected XSS attack. The attack is performed using a single request and response. This attack is possible in web application where the user data inputs are not properly encoded or the special characters used in scripts are not properly filtered.

4.1.2 Stored XSS-Cross Site Scripting

Stored XSS attack[6] is a kind of attack in which malicious script is injected into the web application database using methods like injecting in a comment box in a forum or posting data using input fields.In stored XSS attacks, the malicious script is saved in the application's database permanently and hence known as persistent XSS attack. This attack affects all the users loading from the database and hence more dangerous than Reflected XSS in terms of scale. When an end user loads the web application containing Stored XSS scripts, if the data from database is not properly sanitized or filtered the javascript is executed in the browser and sensitive information such as session ids are exposed to the attacker.

4.2 SQL Injection

SQL injection or SQLI[6] is an attack that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed to the normal end user. This database information may include any number of items, including sensitive company data, user lists or user private data.A successful SQLI attack can result in the unauthorized viewing of user lists, the deletion of entire tables in database and, database administrative rights being granted to the end user. An attacker in SQL injection manipulates a standard SQL query to exploit non-validated input(user input fields) vulnerabilities in a database.

Chapter 5

OUR APPROACH FOR MODELLING WEB APPLICATIONS FOR SECURITY TESTING

In this chapter, we discuss our modelling techniques for assessing security for web applications. Creation of a model of a web application requires traversing all paths from the home page of web application to all links present in the web page. Transitions from a web page happens due to static hyperlinks and dynamic javascript events in a webpage(ex. onclick() events). Model of a web application security testing needs to capture all the elements in a webpage and all transitions possible from the webpage. The following sections illustrate our approach to model the web application.

5.1 Document Object Model(DOM) Tree

The Document Object Model (DOM) is a programming interface(API) for HTML and XML documents. It represents the document structure, style and content of web applications. DOM is represented as nodes and objects in the document. A Web page is a document encoded in HTML(XML). This document is displayed in the browser window as an application. The DOM is an object-oriented representation of the web page, which can be modified in the web applications with JavaScript.In DOM representation, every HTML element is considered as an object. The DOM represents the web page as a tree structure of HTML tags.

An example DOM Tree is given in the below figure.

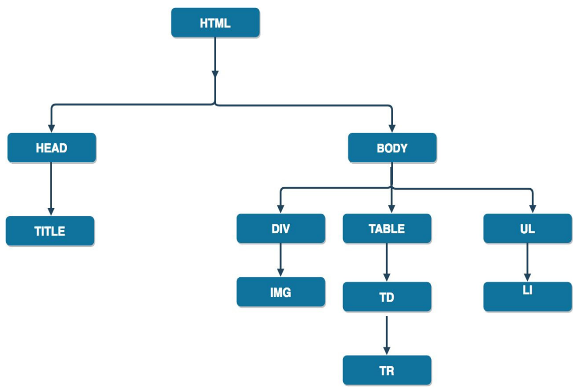


Figure 5.1: Example of DOM Tree

5.2 Navigation Graph Model

In Navigation Graph model[5], a graph is built with nodes and edges where each node represents a Web page and each edge represents a link. It is a model built by a Web crawler (or Web spider) program that automatically traverses the Web application's hyperlink structure and retrieves the content of the Web pages. This model considers only static HTML elements such as hyperlinks and ignore all the dynamic elements such as AJAX elements. In this model, each web page url is a different node and if multiple web pages have the same url(dynamic content) this model does not represent the accurate website structure.

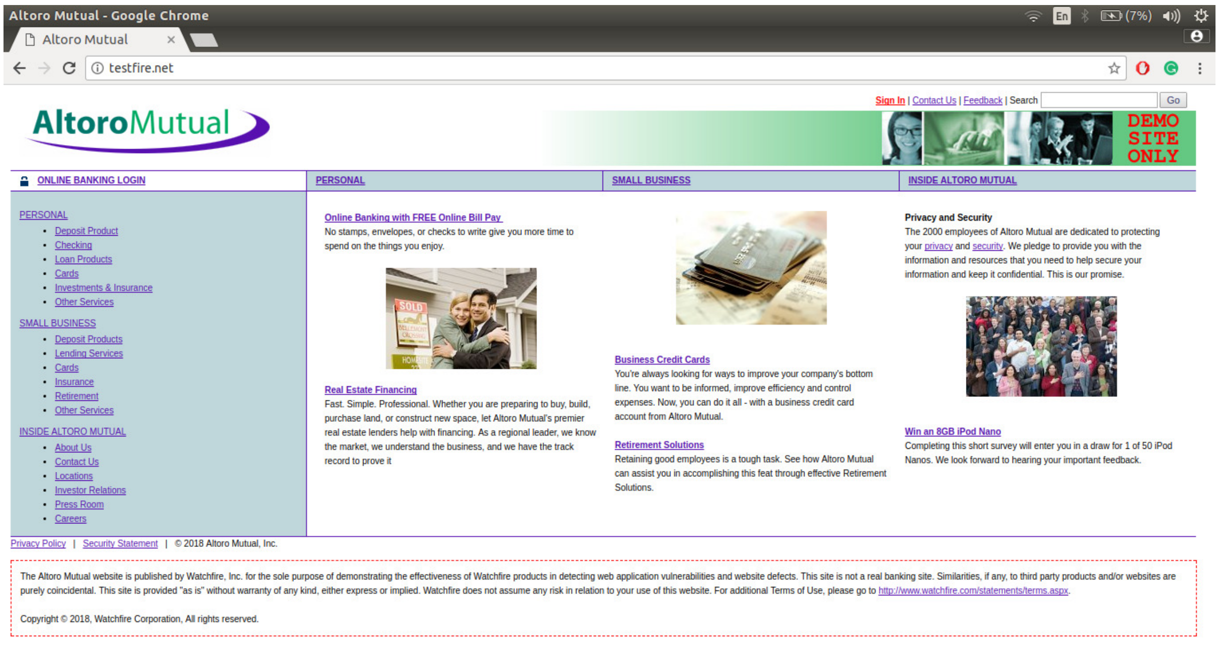
5.3 Finite State Machine

In Finite State Machine, since each node represents a different State of an AJAX web page and each edge between nodes represents a transition performed on a clickable element. In a webpage, each element of the page can became clickable at runtime(dynamic). The home web page is defined as the root or start State and new States are added as the application is crawled. To find the clickable elements in a webpage, the DOM model of the web page is extracted and the clickable elements are obtained. These elements are subjected to click events and then the resulting DOM model is obtained. In case there is a change in DOM, then a new State is created and an edge is added between the States. This crawling procedure is recursively called to find all possible States.

The below figure illustrates a example web application and its corresponding State machine model.

5.4 Example of a web application and its state machine

The below figure represents the web application http://www.testfire.net/



Sign In | Contact Us | Feedback | Search Go

ONLINE BANKING LOGIN | PERSONAL | SMALL BUSINESS | INSIDE ALTORO MUTUAL

PERSONAL
- Deposit Product
- Checking
- Loan Products
- Cards
- Investments & Insurance
- Other Services

SMALL BUSINESS
- Deposit Products
- Lending Services
- Cards
- Insurance
- Retirement
- Other Services

INSIDE ALTORO MUTUAL
- About Us
- Contact Us
- Locations
- Investor Relations
- Press Room
- Careers

Online Banking with FREE Online Bill Pay
No stamps, envelopes, or checks to write give you more time to spend on the things you enjoy.

Real Estate Financing
Fast. Simple. Professional. Whether you are preparing to buy, build, purchase land, or construct new space, let Altoro Mutual's premier real estate lenders help with financing. As a regional leader, we know the market, we understand the business, and we have the track record to prove it

Business Credit Cards
You're always looking for ways to improve your company's bottom line. You want to be informed, improve efficiency and control expenses. Now, you can do it all - with a business credit card account from Altoro Mutual.

Retirement Solutions
Retaining good employees is a tough task. See how Altoro Mutual can assist you in accomplishing this feat through effective Retirement Solutions.

Privacy and Security
The 2000 employees of Altoro Mutual are dedicated to protecting your privacy and security. We pledge to provide you with the information and resources that you need to help secure your information and keep it confidential. This is our promise.

Win an 8GB iPod Nano
Completing this short survey will enter you in a draw for 1 of 50 iPod Nanos. We look forward to hearing your important feedback.

Privacy Policy | Security Statement | © 2018 Altoro Mutual, Inc.

The Altoro Mutual website is published by Watchfire, Inc. for the sole purpose of demonstrating the effectiveness of Watchfire products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. Watchfire does not assume any risk in relation to your use of this website. For additional Terms of Use, please go to http://www.watchfire.com/statements/terms.aspx.

Copyright © 2018, Watchfire Corporation, All rights reserved.

Figure 5.2: A example banking web application

5.4.1 Crawl Level 1

The below figure is a state machine model for web application http://www.testfire.net/ with crawl level 1.

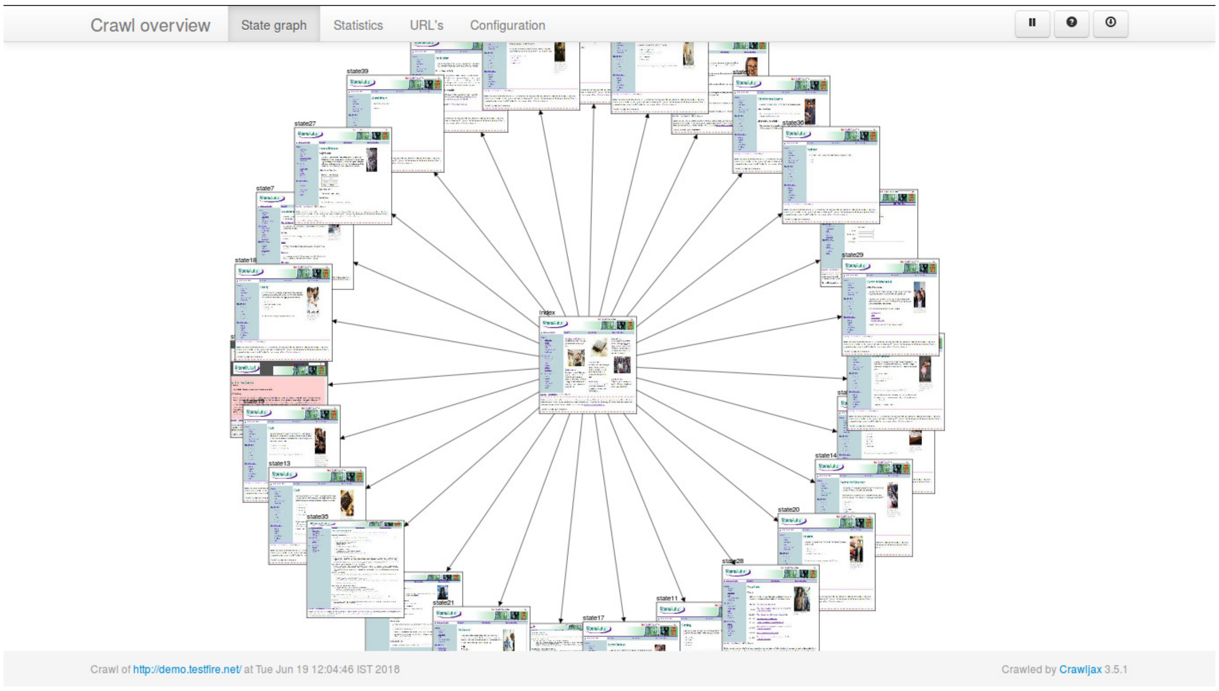


Figure 5.3: State Machine Model for Crawl Level 1

5.5 Choice of the approach used in the dissertation

Since the model web applications are AJAX based and dynamic in nature, the Finite State Machine Model is better than navigation graph with each node representing the State of DOM. Crawljax is selected as the the crawling tool for modelling web applications.

Chapter 6

MODELLING OF WEB APPLICATIONS VIA STATE MACHINE

6.1 Generation of State Machine Model

State Machine for a Web Applications is characterized with a three tuple (r, V, E)

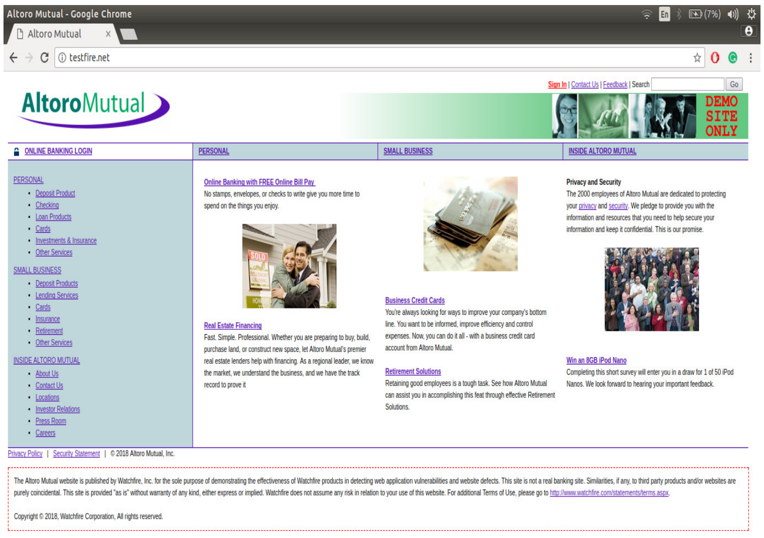
- V is the list of vertices which is recognized as the DOM States in the model
- E is the edges or transitions associated with States after clicking capable events(candidate elements).
- r is the root element or initial State of the model

Crawljax, a open source tool is used to generate the State machine model for web applications. It uses selenium webdriver as a event handler to make direct calls to the browser. The url of web application is the input to create State machine. The browser is intialized and the DOM tree of the web application is extracted and the DOM State is created as the root element. The next step is to find the clickable events in the web-page, both static elements(HTML hyperlinks) and javascript events(eg. Onclick events). These elements are determined as candidate elements for firing transitions in the webpage. Candidate elements are activated one by one and the resulting DOM tree is extracted. The resulting DOM tree is compared with the existing DOM States by the use of a comparator. The DOM tree is converted into strings and the string is the matched with the result State is added to the State machine with an edge added to the previous State. The candidate element from which the transition is done is now removed from the list and the previous State is restored. All the States reachable from the initial State is determined by firing events with all candidate events. This crawl procedure is done for a maximum crawl depth as defined initially.

The below figure illustrates a example web application and its corresponding State machine model.

6.2 Example of a Web Application and its Model

The below figure represents the web application http://www.testfire.net/



Altoro Mutual - Google Chrome

Altoro Mutual

testfire.net

Sign In | Contact Us | Feedback | Search Go

AltoroMutual

DEMO SITE ONLY

ONLINE BANKING LOGIN | PERSONAL | SMALL BUSINESS | INSIDE ALTORO MUTUAL

PERSONAL
- Deposit Product
- Checking
- Loan Products
- Cards
- Investments & Insurance
- Other Services

SMALL BUSINESS
- Deposit Products
- Lending Services
- Cards
- Insurance
- Retirement
- Other Services

INSIDE ALTORO MUTUAL
- About Us
- Contact Us
- Locations
- Investor Relations
- Press Room
- Careers

Online Banking with FREE Online Bill Pay
No stamps, envelopes, or checks to write give you more time to spend on the things you enjoy.

Real Estate Financing
Fast. Simple. Professional. Whether you are preparing to buy, build, purchase land, or construct new space, let Altoro Mutual's premier real estate lenders help with financing. As a regional leader, we know the market, we understand the business, and we have the track record to prove it

Business Credit Cards
You're always looking for ways to improve your company's bottom line. You want to be informed, improve efficiency and control expenses. Now, you can do it all - with a business credit card account from Altoro Mutual.

Retirement Solutions
Retaining good employees is a tough task. See how Altoro Mutual can assist you in accomplishing this feat through effective Retirement Solutions.

Privacy and Security
The 2000 employees of Altoro Mutual are dedicated to protecting your privacy and security. We pledge to provide you with the information and resources that you need to help secure your information and keep it confidential. This is our promise.

Win an 8GB iPod Nano
Completing this short survey will enter you in a draw for 1 of 50 iPod Nanos. We look forward to hearing your important feedback.

Privacy Policy | Security Statement | © 2018 Altoro Mutual, Inc.

The Altoro Mutual website is published by Watchfire, Inc. for the sole purpose of demonstrating the effectiveness of Watchfire products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. Watchfire does not assume any risk in relation to your use of this website. For additional Terms of Use, please go to http://www.watchfire.com/statements/terms.aspx.

Copyright © 2018, Watchfire Corporation, All rights reserved.

Figure 6.1: A example banking web application

6.2.1 Crawl Level 1

The below figure is a state machine model for web application http://www.testfire.net/ with crawl level 1. In crawl level1, initial start DOM state is home page of web applications and all the state reachable from the start state is modelled.

6.2.2 Transition between states

Let us consider the transition between states- start state(index page) to state 39.
$"from" : "index", "to" : "state39"$

The transition is caused the event type click through the element submit button having value Go.
$"element" : "Elementnode = [INPUT : null], tag = INPUT,$

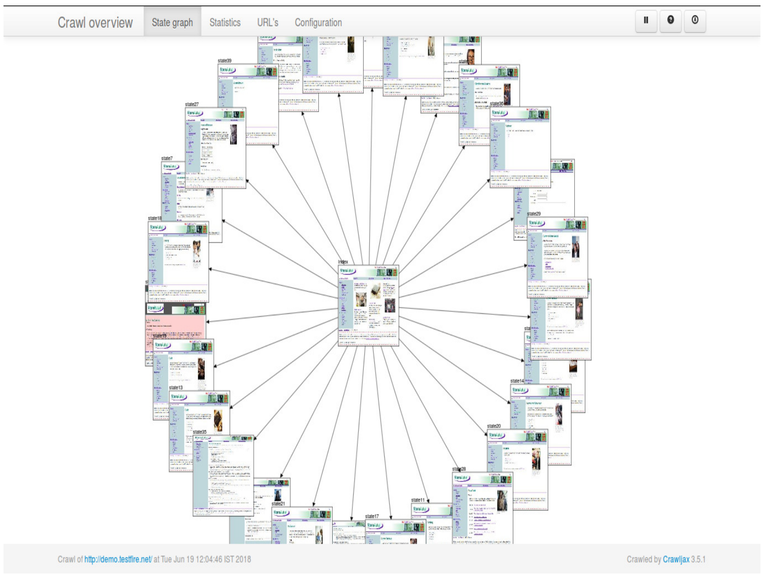

Crawl overview
State graph
Statistics
URL's
Configuration
Crawl of http://demo.testfire.net/ at Tue Jun 19 12:04:46 IST 2018
Crawled by Crawljax 3.5.1

Figure 6.2: State Machine Model for Crawl Level 1

$text =, attributes = type = submit, value = Go", "eventType" : "click"$

The submit button is present in the DOM tree path as show below.
$"id" : "xpath/HTML[1]/BODY[1]/DIV[1]/FORM[1]/TABLE[1]/TBODY[1]/$
$TR[1]/TD[2]/INPUT[2]",$

Similarly, the transition between the states- (index page) to state35 can be explained as below. $"from" : "index", "to" : "state35",$

The transition is caused by selecting a hyperlink default.aspx?content=security.htm
$"element" : "Elementnode = [A : null], tag = A, text = security,$
$attributes = href = default.aspx?content = security.htm", "eventType" : "click"$

The hyperlink is present in the DOM tree path below
$"id" : "xpath/HTML[1]/BODY[1]/DIV[2]/TABLE[1]/TBODY[1]/$
$TR[2]/TD[2]/SPAN[1]/TABLE[1]/TBODY[1]/TR[1]/TD[3]/A[2]",$

6.2.3 Crawl Level 2

In crawl level 2, the states reachable from start state are captured and the modelling procedure is repeated with these states as starting states unlike crawl level 2. The below figure is a State Machine Model derived from web applications http://www.testfire.net for a crawling level of 2.

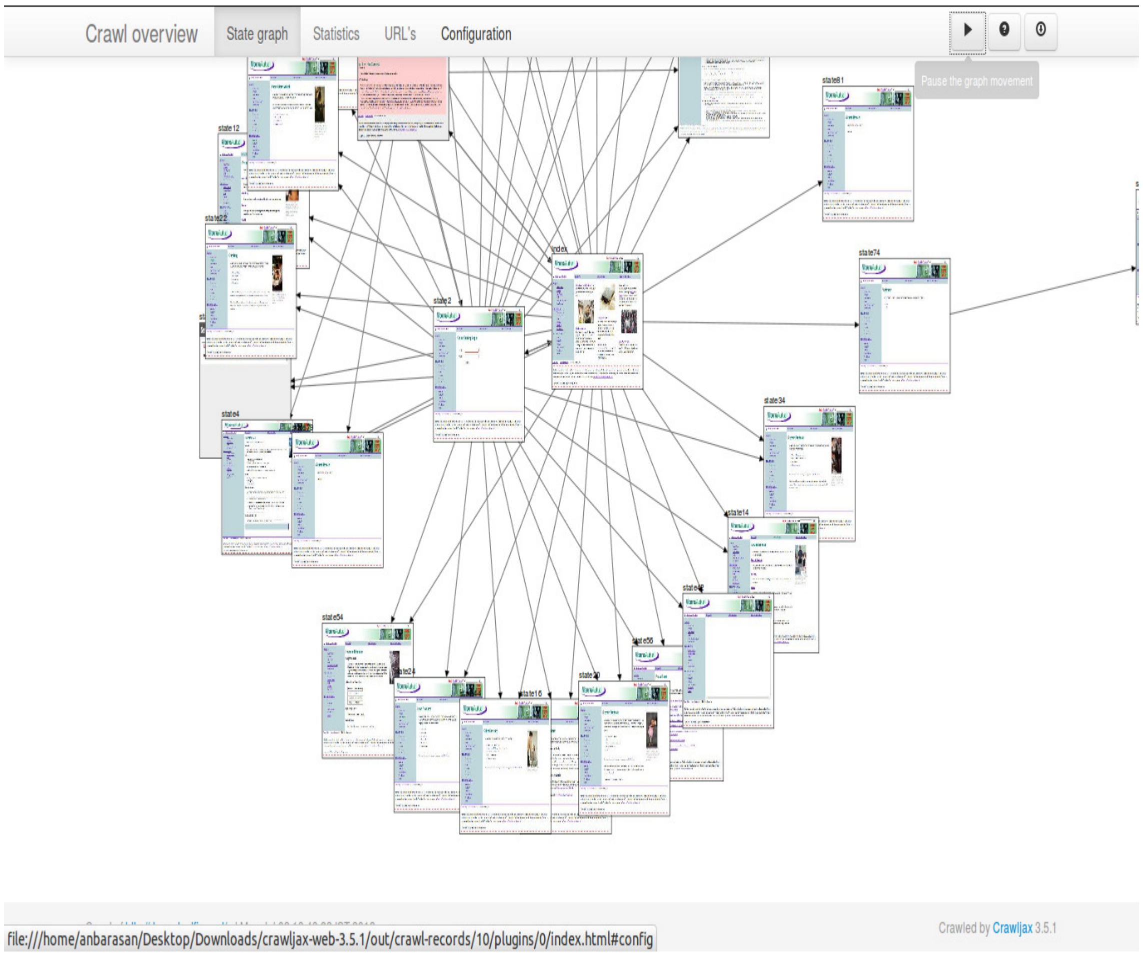


Figure 6.3: State Machine Model for Crawl Level 2

Chapter 7

SECURITY TEST FRAMEWORK

Architecture for Security test framework is depicted in Figure 7.1. It has been implemented in python using requests library and mechanize browser[8] through an API. Requests are used to send http requests and acquire responses from web applications. Mechanize is a Stateful and headless web browser. Headless web browser or UI less web browser is suited for our purpose because in some cases of browsers with UI like chrome or firefox, browser side security can prevent the security attacks while the web application is still vulnerable to security. Such kind of attacks is made possible in headless browser and it is identified in our experimentation.

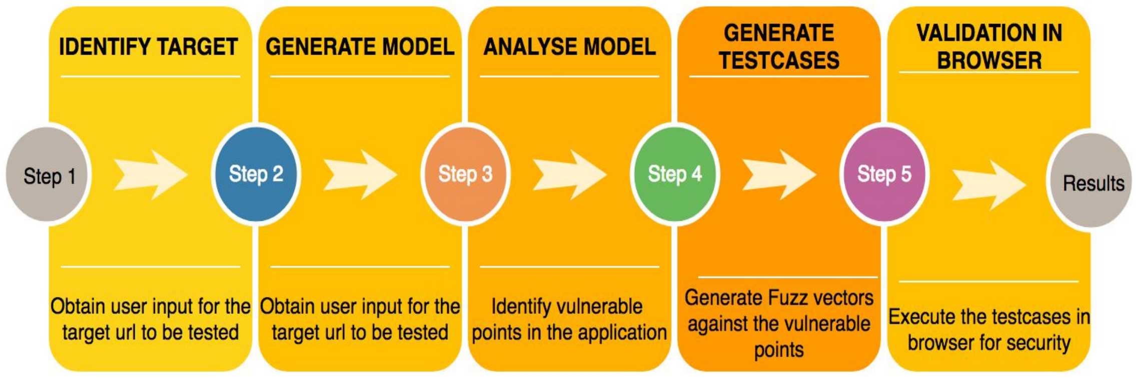

IDENTIFY TARGET
GENERATE MODEL
ANALYSE MODEL
GENERATE TESTCASES
VALIDATION IN BROWSER
Step 1
Step 2
Step 3
Step 4
Step 5
Results
Obtain user input for the target url to be tested
Obtain user input for the target url to be tested
Identify vulnerable points in the application
Generate Fuzz vectors against the vulnerable points
Execute the testcases in browser for security

Figure 7.1: Architecture

7.1 Identification of Vulnerable points of attack from the model

The State machine model represents the architecture and the DOM elements of the the web applications. The next step is to define the attack pattern and test objective of

Security Testing. The most vulnerable security threats are XSS and SQLI attacks. The candidate elements such as forms fields, user inputs, GET and POST parameters which are vulnerable to these attacks are determined from the the DOM tree extracted in the State Machine model as follows.
DOM tree of the start state is obtained from the state machine. Tags in the DOM Tree are parsed only by one and user modifiable elements are captured. Form fields, user inputs, GET/POST parameters and textarea fields, are identified added to a list named as vulnerable elements. Form fields are identified with their tag attribute id. Both the form name and id attributes are added to the list. User Input fields, password field are identified with their tag attribute name. If tag attribute id is not present then tag attribute name is used for input fields identification. Input Elements whose element those are hidden in the web page are also added to the vulnerable elements. All the fields identified as vulnerable elements are stored in a file Vulnerability_model. If there are no user modifiable fields, then there are no candidate elements are found for vulnerability for that particular DOM state. The DOM state is marked as visited and the process is repeated for all states in the state machine.. The vulnerability_model is output to the user as HTML report for all the states. This analysis is implemented by a parsing the DOM Tree using Beautifulsoup library[9].

Example of Vulnerable fields identified from model;

$< inputclass = ''loginInput''name = ''username''size = ''20''type = ''text''/ >$
$< inputautocomplete = ''off''class = ''loginInput''name = ''password''size = ''20''type = ''password''/ >$
$< inputname = ''Login''type = ''submit''value = ''Login''/ >$

7.2 Test Suite Generation based on the Model

The vulnerability points are tested for XSS Attacks and SQL Injection attacks. The test cases are generated for these vulnerability points using the following methods.

7.2.1 Test Suite Generation for XSS Attack

Consider a simple model with a webpage having a input element such a search box and a submit button which displays the results of a search query.

The webpage http://www.xssgame.com/f/m4KKGHi2rVUN/ is show as below:
The test case for XSS attack is derived from the model as follows.
State Tested: Start State
Target Element: Search Input Box (identified from model)
$< inputid = ''demo2 - query''name = ''query''onfocus = ''this.value ='' ''value = ''Enterqueryhere...''/ >$
$< inputid = ''demo2 - query''name = ''query''onfocus = ''this.value ='' ''value = ''Enterqueryhere...''/ >$

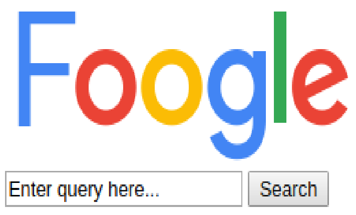


Figure 7.2: Test Webpage for XSS Attack

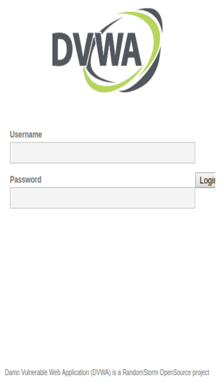
Attack Script: $< script > alert('XSS') < /script >$

If the XSS script gets reflected as $< SCRIPT > alert('XSS'); < /SCRIPT >$ in HTTP Response then the attack has succeeded. This implies there is no filters or input sanitization performed for the attack script. The javascript is executed by the browser.
Instead, it it gets displayed as$lt; SCRIPTgt; alert('XSS') < /SCRIPTgt;$ then the attack is not successful, since the special characters of script are escaped and encoded. The javascript will not be executed by the browser. Similarly the test cases are generated for multiple states, target elements and attacks scripts. The testcases are variants of javascript, execution of javascript embedded in different tags like body, img src. Javascript codes to escape the filters of web application are also part of test cases.
There are two variants of XSS attacks, Persistent or Stored XSS and Non-persistent or Reflected XSS attacks.
In Non-persistent or reflected XSS attack, the javascript is injected to the user accessing web application but the script is not stored in the web application permanently. For this kind of attack after injecting the code, the resulting HTTP response is verified.
In persistent of stored XSS attacks, the javascript is stored permanently in the database of the web application and the script is executed for all persons who access the web application. For persistent XSS attack, after injecting the script, the HTTP response should be verified and also web application should be loaded again and checked whether the malicious script is loaded. This kind of attack happens when image comment is stored in a web application, or a message is posted in a forum.
Example XSS Attack testcases.
$< BODYONLOAD = alert('XSS') >$
$< IMGSRC = "javascript : alert('XSS')"$
$< /xmp >< script > alert(`XSS') < /script >< xmp >$
$< SCRIPT/SRC = "http : //xss.rocks/xss.js" >< /SCRIPT >$

7.2.2 Test Suite Generation for SQLI Attack

The objective of SQLI inject testing is to read unauthorized stored data from a database at the back end of web application such as passwords and usernames. The test suite for SQLI attack is generated by deriving SQL queries for retrieving stored data. The first phase is to find which all input parameters which will flow into the SQL query Statements. These are the SQL queries which are vulnerable to SQLI attacks. Example of input fields are username and password field which which will be translated as a SQL query to validate credentials in the database. The target input fields vulnerable to SQLI attacks are extracted from the State graph model.

Consider a web page having input elements such a username and password fields.



Damn Vulnerable Web Application (DVWA) is a RandomStorm OpenSource project

Figure 7.3: Test Webpage for SQLI Attack

The sample test case for SQL Injection attack is as follows.
State Tested: Root DOM State (identified from state machine)
Target Element: password Field (identified from state machine)
$< inputclass = "loginInput" name = "username" size = "20" type = "text" / >$
$< inputautocomplete = "off" class = "loginInput" name = "password" size = "20" type = "password" / >$
Attack SQL Query: 1 OR 2=2

Entering a username and password into the query will flow into the database as USERNAME="uname" and PASSWORD="pwd". With the attack SQL Query 1 OR 2=2. SQL Query will flow as USERNAME="uname" and PASSWORD="1" OR 2=2. If the web application does not have security measures to control the attack the password check will be bypassed and application will login. If the resulting response contains invalid password then the attack is successful else the password check is bypassed and attack is successful.

Similarly the test cases are generated for all the states, target elements and SQL Injection queries.

Example SQL Injection Attack testcases.
$`TEST$
$SELECTVERSION();$
$1'OR'1' =' 1$
$1'1$
$1AND1 = 1$
$1'AND1 = (SELECTCOUNT(*)FROMtablenames); --$
$1'ANDnon_existant_table =' 1$
$ORusernameISNOTNULLORusername ='$

7.3 Fuzzing Based Techniques for Test Suite Generation

Fuzz Testing is a software testing technique in which invalid or random generated data called fuzz is created. In our tool, the attack vectors are generated using The steps for fuzz testing include the basic testing steps. The steps for fuzz testing include the basic testing steps-

- Identify the target DOM State.
- Identify GET/POST parameters and user input fields vulnerable to security flaws
- Generate Fuzzed attack vectors using recursive fuzzing, bruteforce fuzzing and replacive fuzzing techniques.

We employ replacive fuzzing by replacing the special characters which can be escaped by their hexadecimal codes. Replacive fuzzing is employed by changing the user modifiable parameters reflected in the URLs of states in the state machine.

7.4 Validation of the attack pattern from model in Browser

The target parameters and testcases are derived from the model as explained in above sections.
Target parameters:

$< inputid = "demo2 - query"name = "query"onfocus = "this.value ='' "value = "Enterqueryhere..."/ >< /xmp >$
$< inputid = "demo2 - query"name = "query"onfocus = "this.value ='' "value = "Enterqueryhere..."/ >$
$< inputclass = "loginInput"name = "username"size = "20"type = "text"/ >< inputautocomplete = "off"class = "loginInput"name = "password"size = "20"type = "password"/ >$

XSS Attack testcases.
$< BODYONLOAD = alert('XSS') >$
$< IMGSRC = "javascript : alert('XSS')"$
$< /xmp >< script > alert(`XSS') < /script >< xmp >$
$< SCRIPT/SRC = "http : //xss.rocks/xss.js" >< /SCRIPT >$

SQL Injection Attack testcases.
$`TEST$
$SELECTVERSION();$
$1'OR'1' =' 1$
$1'1$
$1AND1 = 1$
$1'AND1 = (SELECTCOUNT(*)FROMtablenames); --$
$1'ANDnon_existant_table =' 1$
$ORusernameISNOTNULLORusername ='$

The execution framework is implemented in python using Requests Library and Mechanize browser. Mechanize is a headless browser, browser without UI and it is suited for our purpose because browsers like firefox, chrome has security mechanisms implemented with them and they can block attack scripts and provide false negatives. But the web application may be still vulnerable to security attacks. Requests library is used to load content from the web application, the requested content is intercepted. The target parameters identified from the model are now injected with malicious scripts from testcases in the intercepted content. Browser object is created for mechanize browser. The intercepted content is now submitted to the mechanize browser using the browser object created.

The resulting HTTP response from the web application is stored in a debug.log file. The resulting HTTP response from the debug.log file is now analysed whether the injected script is executed. If the HTTP Response contains the injected javascript without filters then it implies the reflected XSS attack is possible. If the elements cannot be modified, the response is obtained as the element is read only. This implies the attack is not successful. For stored XSS attack, after injecting the script, open the browser web application again through a different browser object and find whether the web application contains the injected code. For SQL Injection attacks, if the resulting HTTP response contains data from the web application database, the attack is successful. For malformed SQL queries, if the HTTP response contains any syntax error or database error or database version is obtained, the attack is successful since the SQL query gets executed is not filtered by the web application. If the SQL query is not executed and if the HTTP response contains invalid input, attack is not successful.

The output obtained is written into a summary file for all the states whether the attack is successful or not.

In the next chapter, we shall illustrate our framework with a CASE study.

Chapter 8

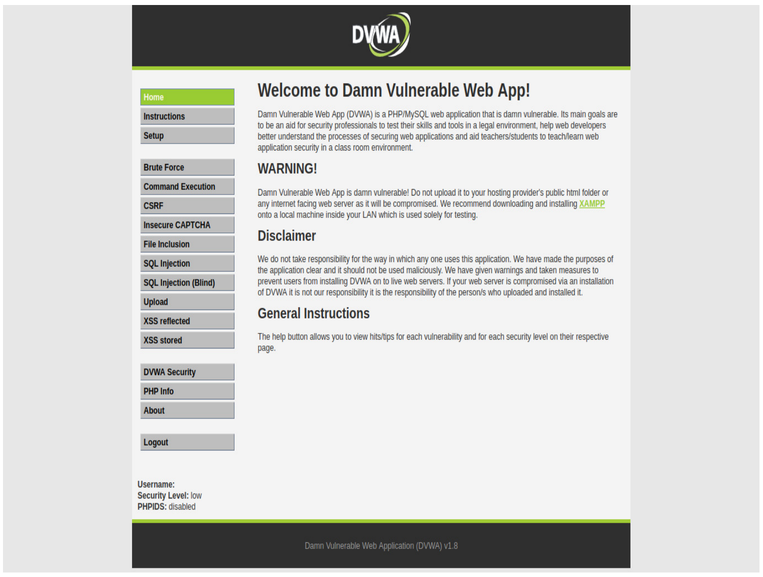
EXPERIMENTAL RESULTS

In this chapter, we shall highlight the results of security vulnerabilities on various web applications.

8.1 Case Study Coverage- DVWA Web Application

Let us consider a case study of web application named Damn Web Vulnerable Web Application for our Model Based Security Testing Approach.

DVWA-Damn Web Vulnerable Application is a PHP/MySQL application which contains security vulnerabilities such as SQL Injection, Reflected XSS and Stored XSS Attacks. There are three security levels - low, medium and high with decreasing order or security vulnerabilities across the levels. The framework is tested against DVWA tools across the security levels.
The below figure depicts the Damn Vulnerable Web Application



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

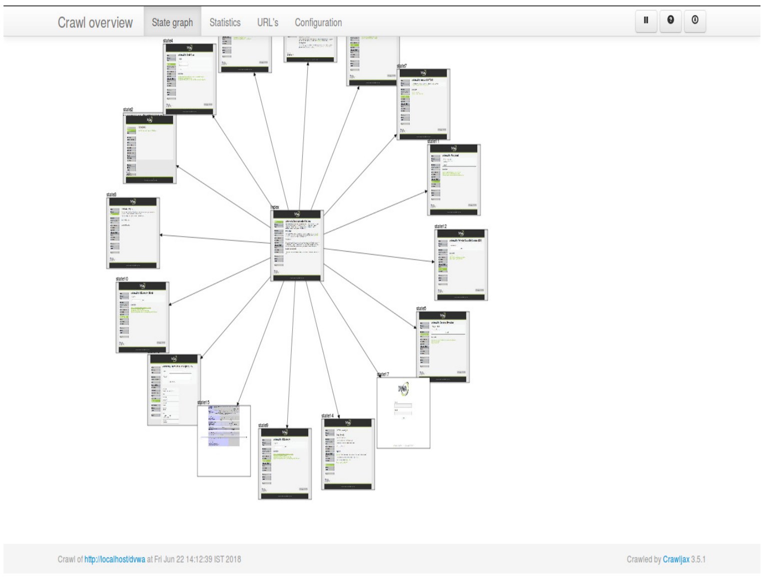
Username:
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8

Figure 8.1: State Machine Model of DVWA

8.2 State Machine Model of DVWA web application

The first phase of our case study is to generate a state machine model for Web application. The Figure 8.1 illustrates the state machine model generated for the application DVWA for crawl level 1. Also the number of states and their url is depicted in the Table 8.1



Crawl of http://localhost/dvwa at Fri Jun 22 14:12:39 IST 2018

Crawled by Crawljax 3.5.1

Figure 8.2: State Machine Model of DVWA

Table 8.1: State Generated in Model

/dvwa/vulnerabilities/sqli/	state9,
/dvwa/vulnerabilities/sqli/	state9,
/dvwa/vulnerabilities/csrf/	state6,
/dvwa	index,
/dvwa/vulnerabilities/exec/	state5,
/dvwa/vulnerabilities/fi/?page=include.php	state8,
/dvwa/vulnerabilities/captcha/	state7,
/dvwa/instructions.php	state2,
/dvwa/vulnerabilities/brute/	state4,
/dvwa/setup.php	state3,
/dvwa/vulnerabilities/xss_s/	state13,
/dvwa/vulnerabilities/xss_r/	state12,
/dvwa/vulnerabilities/upload/	state11,
/dvwa/vulnerabilities/sqli_blind/	state10,
/dvwa/login.php	state17,
/dvwa/about.php	state16,
/dvwa/phpinfo.php	state15,
/dvwa/security.php	state14

8.3 Points of vulnerability identified from Model

The second phase of our automated test tool is identifying the points of vulnerability from the model. It is done by analyzing the state machine model and by parsing the DOM Tree for parameters which require user inputs(GET/POST parameters) form fields, hidden inputs, parameters than can be modified. These parameters are identified as target parameters which are vulnerable for security attacks.

Vulnerability fields for XSS and SQL Injection

State9

$< inputname = ”id” type = ”text” / >$

$< inputname = ”Submit” type = ”submit” value = ”Submit” / >$

State17

$< inputclass = ”loginInput” name = ”username” size = ”20” type = ”text” / >$

$< inputautocomplete = ”off” class = ”loginInput” name = ”password” size = ”20” type = ”password” / >$

$< inputname = ”Login” type = ”submit” value = ”Login” / >$

State13

$< inputmaxlength = ”10” name = ”txtName” size = ”30” type = ”text” / >$

$< inputname = ”btnSign” onclick = ”returncheckForm(); ” type = ”submit” value = ”SignGuestbook” / >$

State10

$< inputname = ”id” type = ”text” / >$

$< inputname = ”Submit” type = ”submit” value = ”Submit” / >$

State4

< *inputname* = ”*username*”*type* = ”*text*”/ >
< *inputautocomplete* = ”*none*”*name* = ”*password*”*type* = ”*password*”/ >
< *inputname* = ”*Login*”*type* = ”*submit*”*value* = ”*Login*”/ >
State11
< *inputname* = ”*MAX_FILE_SIZE*”*type* = ”*hidden*”*value* = ”100000”/ >
< *inputname* = ”*uploaded*”*type* = ”*file*”/ >
< *inputname* = ”*Upload*”*type* = ”*submit*”*value* = ”*Upload*”/ >
State7
< *inputname* = ”*step*”*type* = ”*hidden*”*value* = ”1”/ >
< *inputautocomplete* = ”*none*”*name* = ”*password_new*”*type* = ”*password*”/ >
< *inputautocomplete* = ”*none*”*name* = ”*password_conf*”*type* = ”*password*”/ >
< *inputname* = ”*Change*”*type* = ”*submit*”*value* = ”*Change*”/ >
State5
< *inputname* = ”*ip*”*size* = ”30”*type* = ”*text*”/ >
< *inputname* = ”*submit*”*type* = ”*submit*”*value* = ”*submit*”/ >
State6
< *inputautocomplete* = ”*none*”*name* = ”*password_new*”*type* = ”*password*”/ >
< *inputautocomplete* = ”*none*”*name* = ”*password_conf*”*type* = ”*password*”/ >
< *inputname* = ”*Change*”*type* = ”*submit*”*value* = ”*Change*”/ >
State12
< *inputname* = ”*name*”*type* = ”*text*”/ >
< *inputtype* = ”*submit*”*value* = ”*Submit*”/ >
State3
< *inputname* = ”*create_db*”*type* = ”*submit*”*value* = ”*Create/ResetDatabase*”/ >
State14
< *inputname* = ”*seclev_submit*”*type* = ”*submit*”*value* = ”*Submit*”/ >

8.4 Validation Results of execution in Browser

The final phase of our tool is presenting the results of execution to the user in a html report. Below are the results of our tool for XSS and SQL Injection security testing.
XSS and SQL Injection Test Results

State11:
No candidate elements for Reflected and stored XSS vulnerability
No candidate elements for SQL Injection

State10:
No Reflected XSS and stored Vulnerability not found for script
SQL Injection Vulnerability found for payload
$1UNIONALLSELECT1, 2, 3, 4, 5, 6, nameFROMsysObjectsWHERExtype = \acute{\acute{U}}--$

State14:
No Reflected XSS and stored Vulnerability not found for script

SQL Injection Vulnerability found for payload
$1ANDUSER\_NAME() =' usr'$
State9:
No Reflected and stored XSS Vulnerability not found for script
SQL Injection Vulnerability found for payload
$1ANDASCII(LOWER(SUBSTRING((SELECTTOP1nameFROMsysobjects$
$WHERExtype =' U'), 1, 1))) > 116$

State12:
textbfPayload Reflected in Response. Reflected XSS Vulnerability found for script
$< SCRIPTSRC = http : //ha.ckers.org/xss.js >< /SCRIPT >$

State13: Payload loaded from web application. Stored XSS Vulnerability found
$' - alert(3)-'$

8.5 Inference of results from Case Study of DVWA web application

The DVWA application contains security vulnerabilities like XSS and SQL Injection. Our Test Framework identifies the security vulnerabilities of the web application correctly and identifies which parameters are vulnerable which aids for web developers to correct the security flaws. An exhaustive testing of web application with numerous payloads are tedious to execute by manual testing and also hidden parameters can be overlooked in manual testing. Our tool aids in overcoming these drawbacks and our case study demonstrates our tool usage in security testing.

Table 8.2: Summary of Web Application Tested - DVWA

Number of states tested	10
Crawl Level configuration	1
Number of elements identified for testing	30
Number of elements found vulnerable	5
Types of vulnerabilities found	SQLI, Reflected XSS, Stored XSS
Types of vulnerabilities not found	CSRF, File Inclusion

Vulnerabilities other than XSS and SQLI like CSRF, File Inclusion are not found as they are beyond scope of our thesis.

Chapter 9

RELATED WORK

Table 9.1: Comparison Analysis of Security Testing Tools

Features	Our Framework	BioFuzz	MBVT Tool	WebMate
Model of the tool	State Machine	State Machine	UML Diagrams	State Machine
Fuzzing based Technique	Yes	Yes	No	Yes
XSS Testing	Yes	No	Yes	No
SQLI Testing	Yes	Yes	Yes	No
Headless Browser support	Yes	Yes	Yes	No
Automation	Yes	Yes	Yes	Yes

We have compared our approach with Webmate[1], Biofuzz[3], MBVT Tool[2]. Webmate is a functional testing tool based on model based testing approach without security attributes testing. MBVT Tool uses UML Diagrams as a modelling technique which required generation of UML diagrams based on functional specification. Our tool generates a model without functional specifications. BioFuzz is a search based testing tool which tests only SQL Injection attacks without XSS Testing support. Our tool also performs XSS Testing. Table 9.1 illustrates the comparative analysis of our tool with other model based security testing tools. Apart from these tools there are other tools like SQLMap which tests only for SQL Injection attack. All the tools used fuzzing based technique for generation of testcases with the exception of MBVT.

Chapter 10

CONCLUSION

Web application vulnerability scanners generate few false positive and false negative results. This can be avoided by blackbox model based security testing. To model the web application, different types of Web Application modelling techniques is discussed. The State machine model based on DOM tree is chosen as the model for model based testing since it is flexible for modern web applications. Since SQLI and XSS are the most common security threats, these two attack scenarios have been chosen as attack models. The test suite is generated with the architecture model against the attack models. The attack pattern is executed in an automated manner against the web application.

Chapter 11

FUTURE WORK

The security test framework in this thesis focus on the major security threat vulnerabilities-XSS and SQL Injection attacks.The current framework does not consider security attacks such as LDAP Injection, Broken Authentication and session Management. The challenges in detecting these attacks are the session IDs and authorization tokes expires within a time frame for web applications and brute force techniques are not effective in discovering these vulnerabilities. The techniques to detect these vulnerabilites are in future scope. Our scope of executing the security testing is within headless browsers and extending it to browsers with UI by bypassing security configurations of browsers is of future scope.

Chapter 12

REFERENCES

1. WebMate: A Tool for Testing Web 2.0 Applications, Authors: Valentin Dallmeier,Martin Burger, Tobias Orth, Andreas Zeller Journal: JSTools'12, June 13, 2012, China.

2. Model-Based Vulnerability Testing for Web Applications, Conference: Software Testing,Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on Authors: Franck Lebeau, Bruno Legeard, Fabien Peureux.

3. Search-based security testing of web applications. Authors: Julian Thomé, Andreas Zeller, Alessandra Gorla. Conference: SBST 2014 Proceedings of the 7th International Workshop on Search-Based Software Testing.

4. Crawling AJAX by Inferring User Interface State Changes Authors: Arie van Deursen, Ali Mesbah, Engin Bozdag. Conference: Web Engineering, 2008. ICWE '08. Eighth International Conference

5. A Crawljax Based Approach to Exploit Traditional Accessibility Evaluation Tools for AJAX Applications. Authors: F. Ferrucci1 , F. Sarro1 , D. Ronca1 , S. Abrahao2. Journal: Information Technology and Innovation Trends in Organizations, May 2011.

6. Automatic Creation of SQL Injection and Cross-Site Scripting Attacks. Authors: Adam Kiezun , Philip J. Guo , Karthick Jayaraman , Michael D. Ernst

7. A Survey on Model-based Testing Approaches: A Systematic Review. Authors: Arilo C. Dias, Neto Rajesh, Subramanyan Marlon Vieira, Guilherme H. Travassos

8. OWASP Top10 Security vulnerabilities.
   https://www.owasp.org/index.php/Top_10-2017_Top_10

9. Mechanize Browser API Documentation.
   http://mechanize.readthedocs.io/en/latest/browser_api.html

10. Beautiful Soup HTML Parser Documentation.
    https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Acknowledgements

I am thankful to my supervisor Prof. R.K.Shyamasundar for his enormous support and insightful suggestions throughout my project. I am grateful to him for providing me his code base support which was very useful in carrying out my experiments. His insightful suggestions to the various problems that I faced during my project, were not only useful, but also helped me in broadening my basic understanding of project area.

Signature: [illegible]

Anbarasan K

163050071

Date: 29 June 2018