Model Based Security Testing for Web Applications

Anbarasan Kannan(163050071)[1]

[1]Department of Computer Science
IIT Bombay

June, 2018

Outline

Web Applications

- Web Applications are application programs that are stored in remote servers and delivered over the Internet using browser interfaces.
- Web Applications are client-server programs that is accessed over a HTTP/HTTPS connection. The user interface of web applications are built using the interplay of HTML, CSS and JavaScript.
- Examples: e-commerce websites, online forms, content management systems and email programs

Web Application Model

- Systems Under Test(SUT) referes to the web application being modelled and subjected to testing.
- A model is an abstraction or simplification of the behavior of the web application under test(SUT).
- The model represents the architecture of the web application.

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.
- Model of the SUT is built from informal requirements or from the functionality of the SUT.
- 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.
- Using the test model and the test selection criteria, Test cases are derived for model based testing.

Model Based Security Testing

- Model based testing approach can be adapted to test security of web applications.
- Model Based Security Testing involves two steps- Modeling activity to capture the behaviorial 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.
- In our approach, the test purposes is to test Cross Site Scripting(XSS) and SQL Injection vulnerability of web applications using finite state machine model.

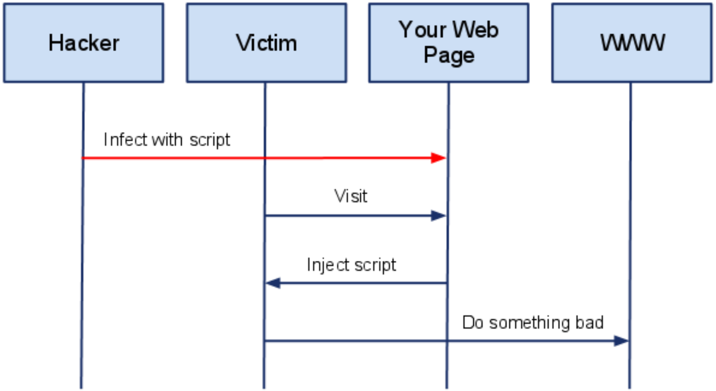
PROBLEM STATEMENT

- 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.
- Web Vulnerability scanners can be used to detect vulnerabilities but they often led to generation of false positive and false negative results.
- Lack of proper documentation of web applications or various technologies used for developing web applications leads to difficulty of deriving model.
- Modern web applications are no more static but dynamic due to the extensive use of Javascript and XML. (Asynchronous Javascript and XML) AJAX. These calls needs to be validated and tested thoroughly to prevent information leaks.

XSS-Cross Site Scripting

- Cross Site scripting or XSS attack targets end-users. User input fields such as form field, url parameter, cookies are vulnerable to XSS attacks.
- An attacker can inject malicious script, usually written in JavaScript, which will be executed by an end-user browser through the input fields in web applications.
- Malicious scripts can access session tokens, cookies, and other sensitive information
- In reflected XSS attacks, the malicious script is executed immediately and response containing malicious data is provided to the end-user.
- In stored XSS attacks, the malicious data is saved in the applications database, and can be retrieved by the attacker.

XSS Attack Pattern

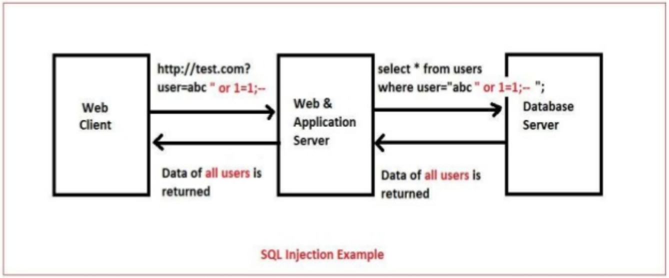


A High Level View of a typical XSS Attack

SQLI-SQL Injection

- SQL injection or SQLI 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.

SQL Injection Attack


Web Client
http://test.com?user=abc " or 1=1;--
Data of all users is returned
Web & Application Server
select * from users where user="abc " or 1=1;-- ";
Data of all users is returned
Database Server
SQL Injection Example

Literature Survey
WebMate: A Tool for Testing Web 2.0 Applications

- Automatically crawls through the Web applications performing regular application testing as well as cross browser compatibility testing.
- 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.

Working of Webmate Testing Tool

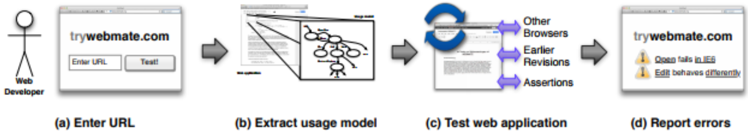


Figure 1: How WEBMATE works. Given a URL as input (a), WEBMATE analyzes the web application and learns a *usage model* of how a user can interact with the web application (b). Users of WEBMATE can then use this model to systematically explore the functionality of the web application and run different analyses such as cross-browser compatibility checks, regression tests or code analyses (c). Results of the analysis can then be reported back to the user (d).

Crawling AJAX-Based Web Applications through Dynamic Analysis

- 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 handler 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.

Search-based security testing of web applications

- Biofuzz, 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 the server.
- Biofuzz tool finds all the input fields with a crawler to detect input fields vulnerable to SQL attacks.

Modelling Techniques for Web Applications

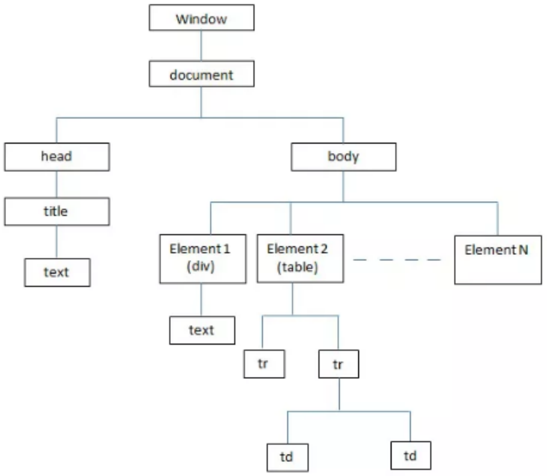
Navigation Graph Model

- In Navigation Graph model, 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 applications 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.

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).
- DOM is an object-oriented representation of the web page, which can be modified in the webpages with JavaScript.
- Every HTML element is considered as an object. The DOM represents the web page as a tree structure of HTML tags.

Example DOM Tree



Finite State Machine

- In Finite State Machine, since each node represents a different state of an web application and each edge between nodes represents a transition performed on a clickable element.
- 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.

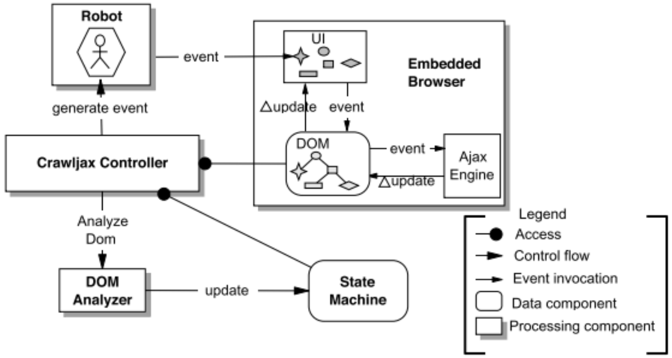
Our Choice of Web Application Model and Tool

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

State Machine Model Structure

- State Machine for a Web Applications is characterized with three tuple (r, V, E)
- r is the root element or initial state of the model
- E is the edges or transitions associated with states after clicking capable events(candidate elements).
- V is the list of vertices which is recognized as the DOM states in the model

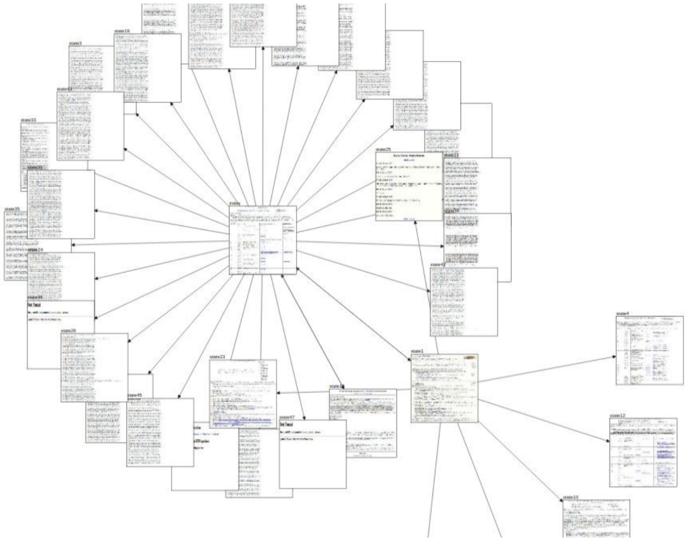
Generation of State Machine Model



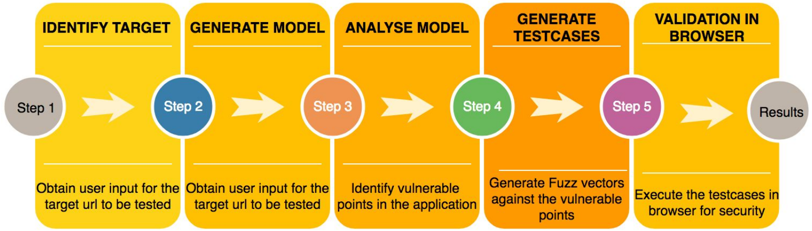
Generation of State Machine Model

- 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 webpage by analyzing DOM Tree, both HTML hyperlinks and javascript events(eg. Onclick events). These are candidate elements for firing transitions.
- Candidate elements are activated one by one and the resulting DOM tree is extracted and 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.
- All the states reachable from the initial state is determined by firing events with all candidate events.

Example State Machine



Security Test Framework


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

Implementation of Test FrameWork

- The Test Automation frameWork is implemented in python using Mechanize Browser
- Mechanize is a headless browser extension supported by python using urllib3 library for HTTP requests.
- The form and input fields are injected with malicious test codes and submitted for each states using request feature in mechanize.
- The HTML response is parsed with BeautifulSoup library in python
- If the security vulnerability is detected, it is reported to the tester in HTML report

Test Suite Generation for XSS Attack

- The test cases are derived for the search query with the expected initial and final states.
- The initial State is Root DOM state
- Target Elements: Input Box and Form Fields
- Expected Target State: Search results DOM state.
- Test Procedure: A malicious javascript is the input value of the testcase. The result of the testcase can be obtained by parsing the HTML code in the result web page.
- If the code is treated as javascript then any malicious script can be executed in the web application. Any DOM state other than expected DOM state or if the application stays in the initial DOM state indicates that the application is vulnerable to XSS attack.
- The testsuite is generated for all DOM states and different javascript attack codes.

Test Suite Generation for SQLI Attack

- The test suite for SQLI attack is generated by deriving SQL queries for retrieving stored data.
- Target Elements: Input fields such as 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.
- The sequence of SQL queries which potentially extracts data from the database or modifies the data are formulated.
- The expected behavior is that the resulting SQL response should contain an error.

Case Study with Web Application DVWA

DVWA-Damn Web Vulnerable Application is a PHP/MySQL application which contains security vulnerabilities such as SQL Injection, Reflected XSS and Stored XSS Attacks.

DVWA Web applications contains three levels of security low, medium and high with decreasing levels of security vulnerability

DVWA application is hosted in a web server and then tested with our framework

States 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

Points of vulnerability identified from model

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" / >
< 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

$< inputname = "name" type = "text" / >$

$< inputtype = "submit" value = "Submit" / >$

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.
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
$1 UNIONALLSELECT 1, 2, 3, 4, 5, 6, nameFROMsysObjects$
WHERE xtype = Ú́--

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
$1ANDASCII(LOWER(SUBSTRING$
$((SELECTTOP1nameFROMsysobjects$
$WHEREExtype =' U'), 1, 1))) > 116$

State12:
Payload 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
$' - alert(3)-'$

Comparison Analysis of Security Testing Tools

Table: 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	No	No
Headless Browser support	Yes	Yes	No	No
Automation	Yes	Yes	Yes	Yes

Demo

Conclusion

- Web application vulnerability scanners generate few false positive and false negative results.
- This can be avoided by blackbox model based security testing.
- 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 XSS and SQL Injection are the most common security threats, they are chosen as the attack model. The test suite is generated with the architecture model against the attack model.
- The attack pattern(testsuite) is executed in an automated manner against the web application.

Features that can be added

- The focus for future work is the extend the framework to other security attacks such as LDAP Injection, Broken Authentication and session Management.
- Extending it to browsers by bypassing security configurations of browsers.
- Integration of Tool to other regression tools by providing API support.

Bibliography I

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

- Franck Lebeau, Bruno Legeard, Fabien Peureux.
  Model-Based Vulnerability Testing for Web Applications
  2013 IEEE Sixth International Conference

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

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

Bibliography II

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

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

THANK YOU