Online Automatic Security Testing on Web Applications with User Sessions

Shanthakumar M [#], Sukumaran S [*], Janarthanam S[#]

[#1]*Erode Arts & Science College,Erode-9*
[2]*Department of Computer Science, Erode Arts & Science College, Erode-9*
[3]*Department of Computer Science, Gobi Arts & Science College, Gobichettipalayam.*

[1]doctorshanth@gmail.com
[3]professorjana@gmail.com
[2]prof_sukumaran@gmail.com

***Abstract*— The continuous use of the Web applications for daily operations by businesses, consumers, and the government has created a great demand for reliable Web but Web applications are a major target of attackers. One promising approach to testing the functionality of Web applications leverages the user session data collected by Web servers. User session-based testing automatically generates test cases based on real user profiles. The key contribution of this paper is the application of concept analysis for clustering user sessions and a set of heuristics for test case selection. However, bridging the gap between an abstract attack trace output by a model-checker and a penetration test on the real web application is still an open issue. Existing incremental concept analysis algorithms are exploited to avoid collecting and maintaining large user-session data sets and to thus provide scalability. The demand for software testing as an online service is on the rise and is influenced by conditions such as the level of domain knowledge needed to effectively test an application, security and pricing as top requirements, cloud computing as the delivery mode and the need for software testers to hone their skills. We present here a methodology for testing web applications starting from a secure model.**

***Keywords*— Software Testing, Web Analysis, User Session based Testing, Internet, Web Metric, Penetration Testing.**

I. INTRODUCTION

Since web applications handle sensitive data, ensuring their security is important and a prerequisite for their use in business contexts. Web applications usually are not monolithic but consist of several distributed components. During the development of the communication protocols and the web components, different tools and programming languages may be used. White-box penetration testing tools usually require that all applications are developed in the same language (e.g., PHP for Ardilla) which is usually not the case in distributed environments. It provides daily operation, maintenance and testing support through web-based browsers, testing frameworks and servers. This model supports a demand-led software testing market by enabling organizations to provide and acquire testing services whenever needed. It envisions an important contribution to the software industry owing to innovations such as Web services and cloud computing that provide new platforms for software testing. Many of the current testing tools address Web usability, performance, and portability issues [1]. For example, link testers navigate a Web site and verify that all hyperlinks refer to valid documents. Form testers create scripts that initialize a form, press each button, and type preset scripts.

The dynamic distributed structure of web applications poses new challenges to building comprehensive test suites. Users interact with application pages, providing inputs that are used to instantiate the server environment variables. These variables are then used to interact with the database backend, retrieving information used to dynamically construct new client pages to be sent to the users [3].

II. RESEARCH METHODOLOGY

One of the most popular ways which most frequented websites use to collect data and information about their websites is through web analytic. This paper presents an approach for achieving scalable user session-based testing of Web applications [2,8]. The key insight is formulating an approach to selecting test cases that is, user sessions for test suite reduction based on clustering logged user sessions by concept analysis. Recently, model-checkers dedicated to security analysis have proved their ability to identify complex attacks on web-based security protocols.

A. Problem Definition

Current black-box penetration testing tools are not really effective due to the weaknesses of the crawling step that misses lots of potential interaction with the user e.g., the output page may depend on the parameters provided by the user and it is hard to guess those parameters in general, for an evaluation of such penetration scanners that show evidence of those weaknesses. Model describing the specification could help black-box penetration testing tools in discovering

vulnerabilities issues. Here a methodology for online automatic testing on web applications starting from a secure model.[4] First, we mutate the model to introduce specific vulnerabilities present in web applications. Then, a model checker outputs attack traces that exploit those vulnerabilities. Next, the attack traces are translated into concrete test cases by using a 2-step mapping. Finally, the tests are executed on the real system using an automatic procedure that may request the help of a test expert from time to time.

B. Test Case Generation

To evaluate our approach in testing, implements a prototype for a Roe Based Access Control (RBAC) and an Cross-Site Scripting (XSS) on an insecure web applications view the collection of logged user sessions as a set of use cases, where a use case is a behaviourally related sequence of events performed by the user through a dialogue with the system [5,9]. Test suites are reduced with the criteria of covering all base requests in the original test suite and covering distinct use cases, where we specify a use case to be a set of base requests. Existing incremental concept analysis techniques can be exploited to analyse the user sessions on the fly as sessions are captured and converted into test cases and, thus, we can continually reflect the set of use cases representing actual executed user behaviour by a reduced test suite.

Each user session is a sequence of user requests in the form of base requests and name-value pairs. When cookies are available, we use cookies to generate user sessions. Otherwise, we say a user session begins when a request from a new Internet Protocol (IP) address arrives at the server and ends when the user leaves the Web site or the session times out. To guide a penetration tester with user sessions when the Test Execution Engine (TEE) needs his help to execute one step of the test cases with performance analysis [7].

III. PROCESS OF RESEARCH

The main contribution using the methodology presented in this paper is the ability to exploit a model describing a Web application at the browser level without low-level communication knowledge to guide a penetration tester in finding attacks based on logical vulnerabilities (e.g., a missing check in a RBAC system, non-sanitized data leading to XSS attacks). To transform a user session into a test case, each logged request is changed into an HTTP request that can be sent to a Web server. Test case consists of a set of HTTP requests that are associated with each user session. Elbaum et al. [10] provided promising results that demonstrate the fault detection capabilities and cost effectiveness of user-session-based testing. Their user session- based techniques discovered certain types of faults; however, faults associated with rarely entered data were not detected. In addition, they observed that the effectiveness of user-session-based testing improves as the number of collected session's increases; however, the cost of collecting, analysing, and replaying test cases also increases [6].

A. Mutation on to real vulnerabilities in Web Applications

To evaluate the performance of Web testing techniques with respect to the detection of faults. Faults were not available with our subject application; thus, to obtain them, we followed a fault seeding procedure similar to one defined and employed in previous studies of testing techniques [1], [11], provide a few mutation operators that reflect the potential presence of specific vulnerabilities and allow a model-checker to generate attack traces that exploit those vulnerabilities.

B. Web Intermediate Application Abstract Processing

Coherent model that embraces effective and efficient testing of multi-tier web applications. inter-connection information across all the tiers in web The methodology first determines the sequences of web applications to depict both intra- and inter-tier dependence pages to be visited that are potentially fault sensitive relationships.[12] The model has been used to facilitate testing Instantiation of abstract attack traces into executable test cases is done by using a 2-step mapping approach. First, the message-based attack trace is mapped to an intermediate language for Web applications, intermediate language is independent from the web application under test, test experts do not need to provide the second mapping, from Web Intermediate Application Abstract (WIAA) to source code.

C. Automatic Test Execution Engine

Finding errors in the tested object and giving confidence in its correct behaviour by executing the tested object with selected input values. Web applications typically undergo maintenance at a faster rate than other software systems and this maintenance often consists of small incremental changes [13]. To accommodate such changes, Web testing approaches must be automatable and test suites must be adaptable. However, Web applications raise important and challenging test issues that cannot be solved directly by existing test techniques for conventional programs.

Automatically executes test cases at the browser level, using the Selenium framework. If an action cannot be performed at the browser level (e.g., an item cannot be selected from a drop-down menu), the Automatic Test Execution Engine (ATEE) may ask for the help of a test expert to provide corresponding HTTP requests; hence the semi-automatic procedure.

A failure can be attributed to any fault in the application implementation. Generally, there will be failures mainly due to faults in the application itself and failures that will be mainly caused by faults in the running environment or in the interface between the application and the environment where it runs Dynamic navigation testing tool for Web applications can explore sequences of links in Web applications by exploring action sequences starting from a given URL can automate the testing process of Web application From the point of view of reverse engineering, our approach can be seen as a reverse engineering approach to model Web application from source code[15,17].

We assume a secure model M as a starting point to the test case generation (i.e., $M \models \phi$ with $\phi \in \Phi$ for a set of interesting security properties Φ). The security properties Φ (e.g., confidentiality, authenticity, authorization) are assumed to be provided together with the secure model.[16] The model of the Web application is built by scanning its source code to identify links and names of input parameters. However, their approach only identifies the names of input parameters and without its domain information, such as parameter types, relevant values of the parameters, so it might not be applied directly to generate the test case. A component cluster is a collection of related components. A computation unit that offers a certain kind of Web service is regarded as a component. In Web applications, a component may be an individual Web page, a software module, or collections of Web pages and software modules.

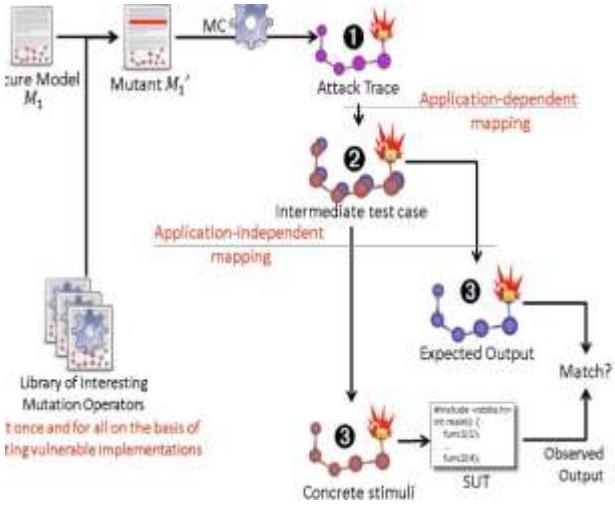

Fig.1 Overview of Test Case Process

Inputs affect the generation of dynamic content and sometimes the target pages that can be navigated, thus affecting page coverage.[18] Though a form could have several types of input fields, we can divide those types into two categories: enumerable and unbounded. Enumerable fields are those fields that in the form definition itself have a list of all possible input values from which the user can choose. Examples of such fields are drop down lists, radio buttons and check boxes. Other types that are slightly different but can be included in this category are buttons and hidden fields since the choices of inputs are limited as well.

Unbounded fields are text based fields that the user has to enter. The possible values are unbounded and therefore are more challenging to automatically generate. Examples of these types are text and password fields and search boxes.[8,19] A form with a larger number of unbounded fields could be considered to have lower testability, because generating meaningful values for these fields may be quite hard. Moreover, these fields could be dependent upon each other.

Fig.2 Test Case Result Expectation Evaluation

Many Web browsers cache page contents, so the users may unknowingly or even deceptively see the contents that are already outdated. The history mechanisms of all major commercial Web browsers are stack-based for Web page navigation, rather than being linear list of visited pages [20].

The actions a user has to perform to access a profile are modelled as follows:

```
Body {% of User % Login
Act ->* S: login(Act, password(Act,S));
S -> Actor: listStaffOf (Act);
% View Profile
if (Act->isAuthorizedToView(?A)) {
Act *-> S: viewProfileOf (A);
S *->* Act: profile Of (A);
}}
```

The mapping of the abstract attack trace to executable source code is a multi-step process because it consists of application dependent and application-independent information is to emulate an execution in which the expected and real application behaviours are compared. One of the most relevant difficulties in web testing is that a lot of manual intervention is often required to fully test the application. In fact by applying only

automatic testing criteria there is no way to guarantee that a web application is 'completely' covered.

IV. EXPERIMENTS AND RESULTS

Defined Agent Frame work get the desired Web Contents, Patterns with abbreviations and acronyms the first Testing the Metrics combined with an original web logs, merging and filtering techniques could be first applied for getting rid of bad and irrelevant entries. Then clustering technique is used for categorizing different types of customers by interested characteristics.

D. Automatic Test Execution

Generating Actions (listed in the tabulation) represent a small but complete set of atomic actions that a user can perform when he uses a web application (e.g., follow a link, click on a button, type text into a text field).

TABLE I
ABSTRACT MESSAGE WITH MODEL

Message	**Abstraction Categories**		
	Model	**WIAP**	**Protocol**
Login	2	5	14
Staff List	1	1	5
View of	1	3	8
Profile of	1	1	2
Search Profile	1	1	3
Attack detect	**1**	1	1

More complex actions can be described by a combination of such atomic actions. For example, log in via a form may correspond to the sequence: select the name from a menu, type the password into a text field, and click on the login button. Since it works at the Browser level, Genetic Algorithm (GAs) are close to API methods from Selenium, a Web application testing framework. However, GAs are not API methods at source code level but abstract browser actions and therefore they are technology independent. For example, after browsing the product list, the customer has 0.3 probabilities to transfer to the Search request. From Browse to next operation, it needs 10 seconds for the think time. User satisfaction is critical for the success of a web site and it comes directly from users' reception. Therefore, metrics from users' perspective could be useful rules for performance testing.It will automatically start the reliability testing process. Firstly, it will perform testing on the validity and currency of the website, in order to checking whether there are data and basic information about the author on the pages. Secondly it will test the status of the hyperlinks .

If the hyperlinks are invalid, this indicates that the website is invalid too. This is why we are validating the hyperlinks of the website as the first step. In the contrast, the mobile agent will traverse the hyperlinks of the website to validate the hyperlinks.

E. Results

The main component that can influence the performance and scalability is the crawling part. The models represent the web application structures and methods used for test case generation with Choosing combinations that represent the typical application behaviour we can encounter is sufficient to achieve high coverage in this case.

Our approach can successfully be applied to all these sessions because the existing vulnerabilities in such sessions can be expressed and described as mutation operators. Nevertheless, there are other lessons (e.g., HTTP splitting attacks) that require details on modelling communication at HTTP level. Since our approach is based on models using high level browser actions, such attacks are not addressed by our approach.

To express the same action, whereas the corresponding HTTP message consists of 13 header lines and 1 content section. The small number of parameters at the model level with regards to the number of parameters at the protocol level is an indication for improved understand ability at the model level.

Nevertheless, there is also solid evidence for the merits of this approach (Grieskamp et al., 2011). The high-level abstraction used by the models in our approach makes it possible to build a secure model from a web application by just using a browser, without looking at the HTTP communication. a dynamic navigation testing tool for Web applications. This tool can explore sequences of links in Web applications by exploring action sequences starting from a given URL and create data for form fields by choosing from a set of name-value pairs provided by the tester. Basically, the tool can automate the testing process of Web application. In this paper, we generate test path with path navigation diagram to model the behaviours of Web application.

The number of parameters in Table I may be misleading. For example, for the profile of message, there is only 1 parameter in both the model abstraction and WAAL, but it corresponds to the user name in the model and to a criterion for distinguishing the profile inside a HTML page, which is more complex than just the user name. In general, parameters at a lower level are more complex to define than parameters at a higher level of abstraction. Then, metrics derived from users' perspective are applied and usage pattern from client side are gained. At last test case can be generated automatically by solving an optimization problem since the high-level and flexible web testing model is essentially a problem, a proper coordination model for

mobile agent may notably simplify the web testing issues

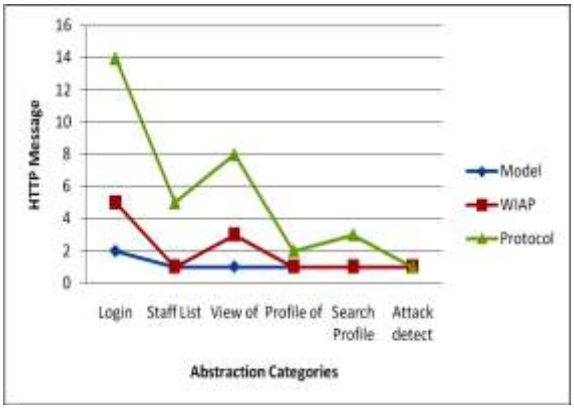

Fig.3. Component Interaction and Abstraction

. It is rare to see the utilization of mobile agent technology on web testing.

V. CONCLUSIONS

The specification is modelled at a high abstraction level (i.e., abstract messages like login, Profile Of or deleteProfileOf, searchProfileOf). Thus, even if our mutants are suitable for generating attack traces, the second challenge is to bridge the gap between the model abstraction and the real implementation to be able to (automatically) execute such attacks on the System under Validation (SUV). We presented a 2-step mapping for this purpose. The idea behind these two mappings is that the second one can be reused for testing other web applications. Thus, we introduced an abstract language (WIAAL) to describe the attack traces at the browser level. The second mapping presented in this paper is suitable for translating actions described in WIAAL to API calls using the Selenium testing framework.

REFERENCES

[1] Doup´e, M. Cova, and G. Vigna, "Why johnny can't pentest:an analysis of black-box web vulnerability scanners," in DIMVA, 2010, pp. 111–131.

[2] Marchetto and P. Tonella, "Using search-based algorithms for ajax event sequence generation during testing," Empirical Softw. Engg., vol. 16, no. 1, pp. 103–140, 2011.

[3] Mesbah, , and A. van Deursen. Invariant-based automatic testing of ajax user interfaces. In 31st International Conference on Software Engineering (ICSE). IEEE Computer Society, May 2009.

[4] F.Dadeau,P.-C.H´eam,and .Kheddam,"Mutation-based test generation from security protocols in HLPSL," in ICST, 2011, pp. 240–248.

[5] G. Fraser, F. Wotawa, and P. Ammann, "Testing with model checkers: a survey," STVR, vol. 19, no. 3, pp. 215–261, 2009.

[6] Kalita, M. 'Investigations on implementation of web applications with different implementation techniques'. Gauhati University,2010

[7] M. B¨uchler, J. Oudinet, and A. Pretschner, "Security mutants for property-based testing," in TAP, 2011, pp. 69–77.

[8] M. Cataldo, A. Mockus, J. A. Roberts, J. D. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures,"IEEE Tran. Software Eng., vol. 35, no. 6, Nov. 2009.

[9] M. Utting and B. Legeard. Practical Model-Based Testing - A Tools Approach. Morgan and Kaufmann, 2006.

[10] M. Wang, J. Yuan, H. Miao, and G. Tan, "A static analysis approach for automatic generating test cases for Web applications," International Conference on Computer Science and Software Engineering, 2008

[11] Martín-Blas, T., & Serrano-Fernández, A. (2009). The role of new technologies in the learning process: Moodle as a teaching tool in Physics. Computers & Education, 52(1), 35–44.

[12] Matthias B¨uchler,Johan Oudinet, Alexander Pretschner, "Semi-Automatic Security Testing of Web Applications from a Secure Model", 2012 IEEE Sixth International Conference on Software Security and ReliabilityISBN:978-0-7695-4742-8/12 pages-253-262

[13] P. Ammann, W. Ding, and D. Xu, "Using a model checker to test safety properties," in ICECCS, 2001, pp. 212–221.

[14] P. Fraternali, G. Rossi, F. Sánchez-Figueroa, Rich Internet Applications, IEEE Internet Computing, vol. 14 nº 3 (2010), pp. 9-12.

[15] Paul M. Duvall, Steve Matyas, and Andrew Glover. Continuous Integration: Improving Software Quality and Reducing Risk. Addison–

[16] Redmill, Felix, "Theory and Practice of Risk-based Testing", Software Testing, Verification and Reliability, Vol. 15, No. 1, March 2005.

[17] S. Artzi, A. Kie˙zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D.Ernst, "Finding bugs in web applications using dynamic test generation and explicit state model checking," IEEE Transactions on Software Engineering, vol. 36, no. 4, pp. 474–494, 2010.

[18] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar,and M. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking,"TSE, vol. 36, no. 4, pp. 474–494, 2010.

[19] S. Sampath, S. Sprenkle, E Gibson, L Pollock, and A Souter,"Appliying concept analysis to user-session-based testing of Web applications," IEEE Transactions on Software Engineering, Page(s):643 – 658, Volume 33, Issue 10, Oct. 2007. Wesley, 2007.

[20] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," TSE, vol. 37, no. 5, pp.649–678, 2011.