e-ISSN: 2349-9745
p-ISSN: 2393-8161
Scientific Journal Impact Factor (SJIF): 1.711
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

Evaluation of Web Security Mechanisms Using Vait

Anantharaman R[1], Hariharan M[2], Mohammed Ansar Ali M.A[3]
[123]*Department of Information Technology, SKP Engineering College, INDIA*

Abstract— This research proposes a methodology and a prototype tool to assess web application security mechanisms. The methodology is based on the inspiration that injecting realistic vulnerabilities in a web application and attacking them automatically can be used to sustain the assessment of presented security mechanisms and tools in custom setup scenarios. To offer true to life results, the projected vulnerability and attack injection approaches relies on the study of a huge number of vulnerabilities in real web applications. In addition to the nonspecific approaches, the manuscript describes the performance of the Vulnerability & Attack Injector Tool (VAIT) that allows the automation of the complete process. We used this tool to run a set of experiments that reveal the possibility and the efficiency of the projected approaches. The experiments comprise the assessment of coverage and false positives of an intrusion detection system for SQL Injection attacks and the assessment of the effectiveness of two top commercial web application vulnerability scanners. Results show that the injection of vulnerabilities and attacks is certainly an effectual way to assess security mechanisms and to point out not only their weaknesses but also ways for their enhancement.

Keywords- security, attacks, VAIT, SQL injection, instrution, vulnerability.

I. INTRODUCTION

Nowadays there is an increasing dependency on web applications, ranging from individuals to large organizations. Almost everything is stored, available or traded on the web. Web applications can be personal websites, blogs, news, social networks, web mails, bank agencies, forums, e-commerce applications, etc. The omnipresence of web applications in our way of life and in our economy is so important that it makes them a natural target for malicious minds that want to exploit this new streak.

Web applications are the most common way to make services and data available on the Internet. Unfortunately, with the increase in the number and complexity of these applications, there has also been an increase in the number and complexity of vulnerabilities. Current techniques to identify security problems in web applications have mostly focused on input validation flaws, such as cross site scripting and SQL injection; with much less attention devoted to application logic vulnerabilities. Application logic vulnerabilities are an important class of defects that are the result of faulty application logic. These vulnerabilities are specific to the functionality of particular web applications, and, thus, they are extremely difficult to characterize and identify. Not astonishingly, the overall circumstances of web application security are quite constructive to attacks investigated in the reports of [1-3]. In the evaluation point to a very huge web appliances with security vulnerabilities investigated in the work of [4], [5] and, accordingly, there are abundant reports of triumphant security breaches and exploitations studied in the research of [6], [7].

Structured crime is clearly affluent in this capable market, if we believe the millions of dollars received by such association in the alternative economy of the web [8-10]. Then, leveraging the knowledge about the typical execution paradigm of web applications, we filter the learned specifications to reduce false positives, and we use model checking over symbolic input to identify program paths that are likely to violate these specifications under specific conditions, indicating the presence of a certain type of web application logic flaws. We developed a tool, called Waler, based on our ideas, and we applied it to a number of web applications, finding previously-unknown logic

vulnerabilities. Clearly, security technology is not good enough to stop web application attacks and practitioners should be concerned with the evaluation and the assurance of their success.

II. MATERIALS AND METHOD

In this section owner or user has to register first, and then only he/she has to right to use the data base. In this unit, any of the above mentioned individual have to login, they should login by giving their email and password. There is disconnecting login for owner and user. The basic block diagram is shown in figure 1.

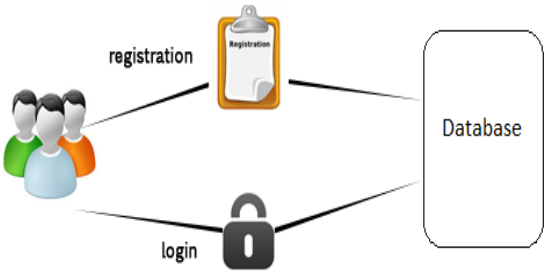

Fig. 1 Basic block diagram.

Use case diagrams replica behaviour explained within a system and helps the developers recognize of what the user necessitate. The stick man characterizes what's called an actor. Use case diagram can be helpful for getting a general view of the system and clarifying that can do and more prominently what they can't do. Use case diagram consists of use cases and actors and shows the communication between the use case and actors.

- The principle is to show the interactions between the use case and actor.
- To symbolize the system necessities from user's perspective.
- An actor could be the end-user of the scheme or an external scheme.

2.1 Vulnerability & Attack Injector Tool:

The projected methodology provides a practical environment that can be used to test counter measure mechanisms (such as intrusion detection schemes (IDSs), web application vulnerability scanners, web application firewalls, static code analyzers, etc.), train and evaluate security teams, help approximation security measures (like the number of vulnerabilities present in the code), among others. This evaluation of security tools can be done online by executing the attack injector while the security tool is also running; or offline by injecting a representative set of vulnerabilities that can be used as a testbed for assess a security tool.

The methodology proposed was implemented in a concrete Vulnerability & Attack Injector Tool (VAIT) for web applications. The tool was tested on top of extensively used applications in two scenarios. The first to assess the effectiveness of the VAIT in generating a large number of realistic vulnerabilities for the offline assessment of security tools, in particular web application vulnerability scanners. The second to show how it can develop injected vulnerabilities to launch attacks, allowing the online assessment of the effectiveness of the counter measure mechanisms installed in the target system, in particular an intrusion detection scheme.

2.2 Attack Injection Methodology:

In this section we present the methodology for testing security mechanisms in the context of web applications. The methodology is based on the injection of realistic vulnerabilities and the

subsequent controlled exploit of those vulnerabilities in order to attack the system. This provides a practical environment that can be used to test counter measure mechanisms (such as IDS, web application vulnerability scanners, firewalls, etc.), train and evaluate security teams, estimate security measures (like the number of vulnerabilities present in the code, in a similar way to defect seeding), among others.

2.3 VAIT internal components:

The complete process is performed automatically, which is explained in figure 2, without human intervention. For example, let's consider the estimate of an IDS: during the attack stage, when the IDS inspects the SQL query sent to the database, the VAIT also monitors the output of the IDS to recognize if the attack has been detected by the IDS or not. We just have to collect the concluding results of the attack injection, which also contains, in this case, the IDS detection output. The automated attack of a web application is a multistage process that includes: preparation stage, attack load generation stage, vulnerability injection stage, attack load generation stage, and attack stage. These stages are described in the next sections.

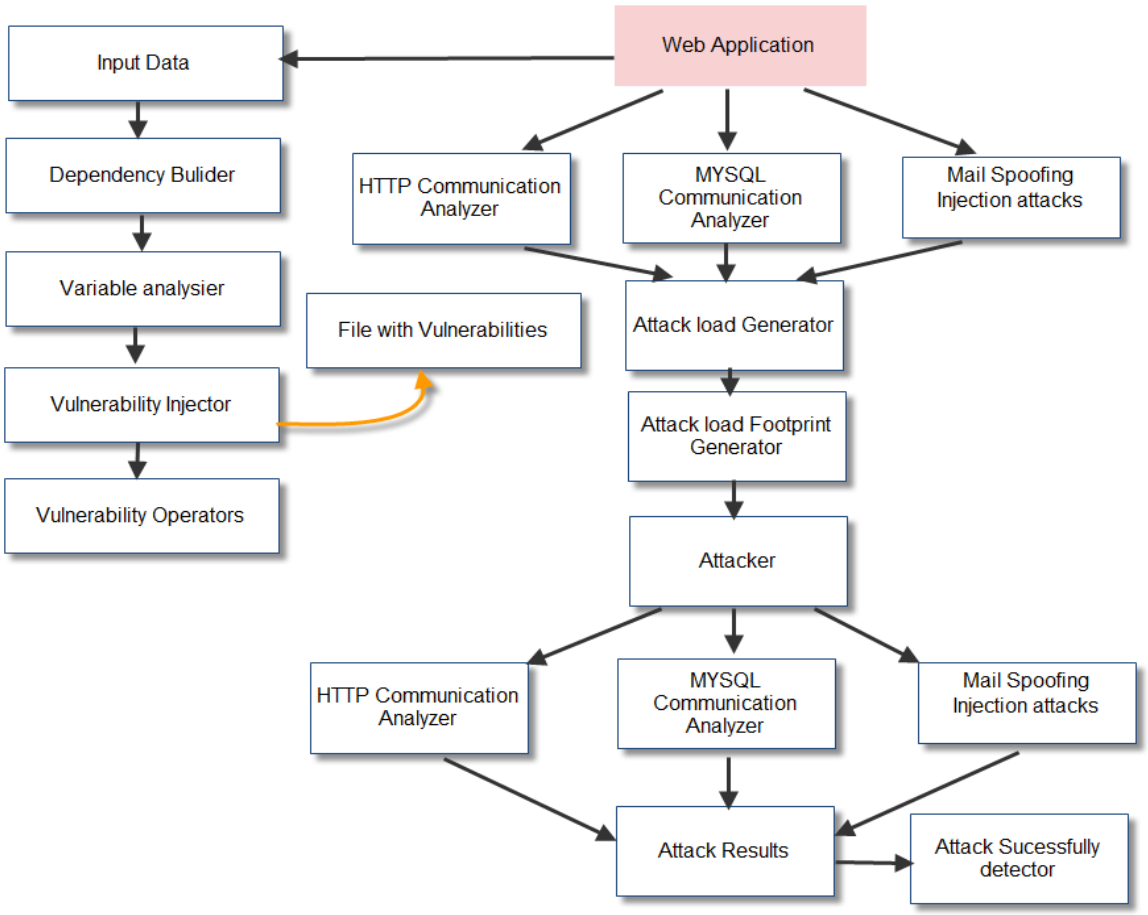

Fig. 2: VAIT Architechture

2.4 Vulnerability Operators

The Vulnerability Operators are built upon a pair of attributes: the Location Pattern and the Vulnerability Code Change. The Location Pattern defines the conditions that a specific vulnerability type must comply with and the Vulnerability Code Change specifies the actions that must be performed to inject this vulnerability, depending on the environment where the vulnerability is going to be injected.

This process of using dynamic and static results provides the best of both worlds to obtain the variables and the location where they are sanitized or filtered and the set of constraints given by the code location required by the Vulnerability Operators.

III. FUTURE SCOPE

It emphasized the need to match the results of the dynamic analysis and the static analysis of the web application and the need to synchronize the outputs of the HTTP and SQL probes, which can be executed as independent processes and in different computers. All these results must create a single analysis log containing both the input and the output interaction results. The VAIT prototype listening carefully on the most important fault type, the MFCE (vulnerabilities caused by a missing function protecting a variable), generating SQLi vulnerabilities. Although this fault type represents the large majority of all the faults classified in the field study and can be considered representative, other fault types can also be implemented, namely those that come next relating to their relevance.

IV. CONCLUSION

A novel technique to automatically inject realistic attacks in web application is presented. This technique consists of analyzing the web application and creates a set of potential vulnerabilities. Each vulnerability is then injected and a variety of attacks are mounted over each one. The success of each attack is robotically assessed and reported. The realism of the vulnerabilities injected derives from the use of the results of a large field study on real security vulnerabilities in extensively used web applications. This is, in fact, a key characteristic of the methodology, because it intends to attack true to life vulnerabilities.

REFERENCES

[1] J. Carreira, H. Madeira, and J.G. Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units," IEEE Trans. Software Eng., vol. 24, no. 2, Feb. 1998.
[2] N. Jovanovic, C. Kruegel, and E. Kirda, "Precise Alias Analysis for Static Detection of Web Application Vulnerabilities," Proc. IEEESymp. Security Privacy, 2006.
[3] IBM Global Technology Services "IBM Internet Security Systems X-Force 2012 Trend & Risk Report," IBM Corp., Mar. 2013.
[4] J. Fonseca, M. Vieira, and H. Madeira, "Training Security Assurance Teams using Vulnerability Injection," Proc. IEEE Pacific Rim Dependable Computing Conf., Dec. 2008.
[5] D. Avresky, J. Arlat, J.C. Laprie, and Y. Crouzet, "Fault Injection for Formal Testing of Fault Tolerance," IEEE Trans. Reliability, vol. 45, no. 3, pp. 443-455, Sept. 1996.
[6] D. Powell and R. Stroud, "Conceptual Model and Architecture of MAFTIA," Project MAFTIA, Deliverable D21, 2003.
[7] V. Krsul, "Software Vulnerability Analysis," PhD thesis, Purdue Univ., West Lafayette, IN 1998.
[8] J. Fonseca and M. Vieira, "Mapping Software Faults with Web Security Vulnerabilities," Proc. IEEE/IFIP Int'l. Conf. Dependable Systems and Networks, June 2008.
[9] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems," IEEE Trans. Computers, vol. 42, no. 8, pp. 913-923, Aug.1993.
[10] J. Duraes and H. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach," IEEE Trans. Software Eng., vol. 32, no. 11, pp, 2013.

