ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

# WEB VULNERABILITY SCANNER USING SOFTWARE FAULT INJECTION TECHNIQUES

S.JEEVA[#1] ,K.RAVEENA[#1], K.SANGEETHA[#1], P.VINOTHINI[#2].
#1 Bachelor of Computer Science and Engineering
#2 Assistant professor - Department of Computer Science and Engineering
Bharathiyar Institute of Engineering for Women

ABSTRACT

We propose a methodology and a prototype to evaluate security in web application.A large amount of web app has been developed. Web applications are typically urban with hard time constraints.There are lot of common weaknesses,it acts as a victim to the business and check this all weaknesses with hand is difficult task. In addition to the some methodology, the paper explains the completion of the Vulnerability & Attack Injector Tool (VAIT) that allows the automation of the whole process. This tool creates feasibility.It is not impossible to compare key figures of web vulnerability scanners. To propose a method to evaluate automatic vulnerability scanners. The impact of a security breach can be very high. The web application market is growing fast, resulting in a huge proliferation of web apps based on different languages frameworks and largely fueled by the simplicity. one can develop and maintain such applications.

KEYWORDS:VAIT,security,review and evaluation.

## 1.INTRODUCTION

To fight this situation we need means to evaluate the security of web applications and of attack contradict measure tools. To handle web application security, fresh tools need to be developed, and events and regulations must be improved, redesigned. further more, everyone involved in the development process should be trained properly. All web applications should be thoroughly validated before going into production. However, these best practices are unfeasible for millions of existing inheritance web applications, so they should be constantly protected by security tools during their lifetime. This is particularly relevant due to the dynamicity of the security scenario, and ways of exploitation being discovered every day. Clearly, security technology is not enough to stop web application attacks and practitioners should be concerned with the assurance of their success . In practice, there is a need for new ways to effectively test existing security mechanisms to improve them.

It proposes a tool to inject weakness and attacks in web applications. The attitude is based on the idea that we can assess different attributes of existing web application security mechanisms and attacking them automatically .It inspired on the failure injection technique that has been used for decades in the dependability area . In our case, the set of "vulnerability" "attack" represents the space of the failure and the "disturbance" is the result of the molest of a "vulnerability" causing to enter in an bad state. A security "vulnerability" is a weakness (an internal "liability") that may be exploited to cause damage, but its presence does not cause harm by itself.
Conceptually, the attack injection consists of the realistic vulnerabilities that are automatically exploited (attacked).Vulnerabilities are considered realistic because they are derived and are injected according to a set of representative restrictions and rules. The attack injection methodology is based on the dynamic analysis is obtained from the runtime monitoring of the web application behavior and of the interaction with external resources, such as the backend database. This information, complemented with the static analysis, allows the effective injection of vulnerabilities. the use of both static and dynamic analysis is a key for the methodology that allows the overall presentation .as it provides the means to inject more vulnerabilities that can be successfully attacked and discarded .
The method can be applied to various types of vulnerabilities, focus on widely exploited and serious vulnerabilities that are (SQLi) and (XSS). Attacks to these vulnerabilities take advantage of improper coded due to unchecked input fields at user interface. It allows the attacker to change the SQL commands or through the input of HTML and scripting languages (XSS).It provides a environment that can be used to test counter measure mechanisms(firewalls, static code analyzer.), train and evaluate security teams, help estimate security measures (like the vulnerabilities current in the code), among others. This evaluation of tools can be done online or offline by injecting are presentative set of vulnerabilities that can be used as a test bed for evaluating a security tool.
A tool to infuse vulnerabilities and attacks in web applications. The proposed methodology is based on the idea that we can evaluate attributes of existing web application security mechanisms by infusing realistic vulnerabilities and attacking them automatically. This follows a procedure inspired on the fault infuse performance. a security "vulnerability" that may be reduce to cause injury, but its presence does not cause injury by itself .
The attack infuse methodology is based on the dynamic analysis of information obtained and of the interface with external resources, such as the backend database. This information, complement with the static breakdown of the source code of the application, allows the effective infuse of vulnerabilities that are similar to those found in the real world. The use of both static and dynamic analysis is a key aspect of the methodology that allows increasing the overall presentation and effectiveness, as it provides the means to infuse more vulnerabilities that can be successfully attacked and eliminated those that cannot. Although this methodology can be applied to different types of vulnerabilities, we focus on two of the most widely broken and serious web application vulnerabilities that are SQL Infuse (SQLi) and Cross Site Scripting(XSS) . Attacks basically take advantage of offensive coded applications due to unchecked input fields at user boundary. This allows the defender to change the SQL commands that are sent to the database .
The methodology proposed was implemented in a real Vulnerability & Attack (VAIT) for web applications. The instrument was tested applications in two scenarios. to estimate the effectiveness of the VAIT first in generating a high number of rational vulnerabilities for

the offline estimation of security tools, in relaxed web application weakness scanners. The second to show how it can exploit infused weakness to launch attacks, allowing the online estimate of the proficiency of the counter measure mechanisms installed in the target system, in meticulous an intrusion system. These experiments illustrate how the proposed methodology can be used in practice, not only to discover existing weaknesses of the tools analyzed, but also to assist improve them.

## 2.RELATED WORK

This paper proposes a methodology and a tool to infuse vulnerabilities and attacks in web applications. The proposed methodology is based on the idea that we can evaluate different attributes of existing web application security mechanisms by infusing realistic vulnerabilities in a web application and attacking them automatically.This follows a procedure inspired on the fault infuse performance that has been used for decades in the fidelity area. . The set of "vulnerability" represents fault space infused in a web application, the"invasion" is the result of the winning of a "attack" causing the application to penetrate in an "fault" state. The attack infuse methodology is based on the dynamic analysis of information obtained from the runtime monitoring of the web application behavior and of the interface with external resources, such as the backend database. This information, complement with the static breakdown of the source code of the application, allows the effective infuse of vulnerabilities that are similar to those found in the real world. The use of both static and dynamic analysis is a key aspect of the methodology that allows increasing the overall presentation and effectiveness, as it provides more vulnerabilities that can be successfully attacked and discarded those that cannot.
Although this methodology can be applied to different types of vulnerabilities, we focus on two of the most widely broken and serious web application vulnerabilities that are SQL shot and (XSS) . Attacks to these vulnerabilities take of offensive coded applications due to unchecked fields of the input at user interface. This allows the defender to change the SQL commands that are sent to the record through the input of HTML and scripting languages (XSS).
The methodology proposed was implemented in a real Vulnerability & Attack (VAIT) for web applications. The instrument was tested on top of widely used applications in two scenarios. The first find thae effectiveness of the VAIT in generating a high number of rational vulnerabilities for the offline estimation of security tools, in relaxed web application weakness scanners. The second to show how it can exploit infused weakness to launch attacks, allowing the online estimate of the efficiency of the counter measure mechanisms copied in the target system, in particular an intrusion system. These experiments illustrate how the proposed methodology can be used in practice ,not only to discover existing weaknesses of the tools analyzed ,but also to assist improve them.

The industry uses fuzzing and Mutation testing penetration testing of web applications. They rely on scanner tools that generate compliant reports with security regulations (Sarbanes-Oxley, PCI-DSS, etc.). Some of the tools are HP Web Inspect, web securify. In spite of their continuous development, these tools still have many problems of undetected vulnerabilities in false positives, To address these problems, it was proposed a method to benchmark these scanners The method starts by identifying all the points where each type of bug can be injected,. Many of these bugs injected that can be used to test and compare the performance of the scanners. The model finders also used for protection analysis.In this case, the vulnerability is injected by mutating the formal model of the web application. The model is also used to generate test cases that are used to attack the web application in a semi-automatic way. The list of possible types of vulnerabilities affecting web applications is enormous, but XSS and SQLi are at the top of that list, accounting for 32 percent of the vulnerabilities observed. SQLi and XSS.

An SQLi attack consists of tweaking the input fields of the webpage (which can be visible or hidden) in order to alter the query sent to the back-end database. These allow the attacker to retrieve sensible data or even alter database records. An SQLi attack can be dormant for a while and only be triggered by a specific event, such as the periodic execution of some procedures in the database (e.g., the scheduled database record cleaning function) A XSS attack consists of injecting HTML and scripting (usually Javascript) in a vulnerable webpage. It exploits the common utilization of the user input (without sanitizing it first) as a building part of a webpage. When this occurs, by tweaking the input, the attacker some of its functions, allowing him to take benefits of users visiting that webpage. This attack exploits user confidence (victim) has on the website, allowing the attacker to impersonate these users and even execute other types of attacks such as cross site request forgery (CSRF) [29]. The injection of XSS can also be persistent if the malicious string is stored in the back-end database of the web application, therefore potentiating its malicious effects in a much broader way.The methodology relies on this paper is the results of the field study presented in [16] to define the types of vulnerabilities to be injected (fault models), which match the most common types of vulnerabilities found in web applications in the field. These vulnerabilities are injected according to a set of representative restrictions and rules previously proposed in [17] and then attacked.

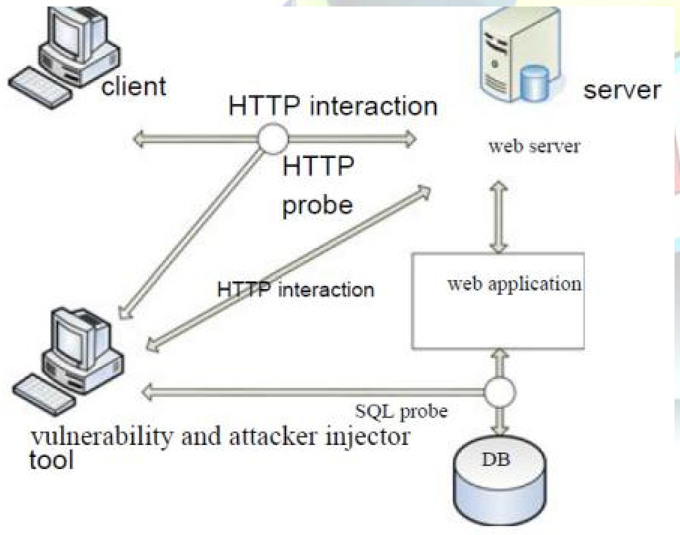
## 3.METHODOLOGY

In this section we present the method for testing security mechanisms for web applications. The method is based on the injection of realistic weakness and the subsequent controlled exploit of those vulnerabilities in order to attack the system. This provides a real time environment that can be used to test counter measure mechanisms (such as IDS, web application vulnerability scanners etc.), train and evaluate teams, estimate security measures (like the number of vulnerabilities in the code, in a same way to defect seeding [31]), compared to others. To provide a realistic environment we must consider true to life vulnerabilities. As mentioned before, we rely on the results from a field study presented in [16] that classified 655 XSS and SQLi security patches of six widely used LAMP web applications. This data allows us to define where a real vulnerability is usually located in the source code and what is the piece of code that is responsible for the presence of such vulnerability.

### 3.1Overview of the Methodology

Our Vulnerability & Attack Injection method for SQLi and XSS can be applied to a variety of technologies, but the upcoming description uses as reference for a web application, with a web front-end and ZUIUX. an access to a back-end database is used to store the dynamic content and business data The vulnerabilities are injected in the web application following some a realistic pattern The information about what was inserted is fed to the injection mechanism mainly for improving the attack success rate .the attack injection uses external explore two: one for the HTTP and other for the database . These probes monitor the HTTP and ..SQL data exchanged, and send a copy by the attack injection mechanism. This is a key factor for this methodology to obtain interaction of the user and the results produced by such a interaction , so they can be used for attack preparing. Therefore, the attack injection mechanism is a important inner workings of the application . For instance, this provides insights on what piece of in order supplied to a HTML FORM is used to build the SQL query and in which question part it is going to be inserted. The whole process is performed automatically, without human work. For example, consider the evaluation of an IDS:, when the IDS inspects the SQL query sent to the database, the VAIT also monitors the IDS output to find if the molest has been detected by the IDS or not. We just have to collect the results of the attack injection, which also contains, the IDS detection output.

The automated attack of a web app is a multiple stage procedure that includes: preparation , vulnerability injection, attack load generation and error.



3.2. Preparation stage:

In the early stage, the web application is interacted (crawled) executing all the functionalities that need fro testing. Both HTTP and SQL languages are captured by the two errors and processed for future use .The interaction with the web application is done from the client's view (the web browser). The outcome of this stage is the relationship of the input and the HTTP variables carry them and their respective source code files, and its use in the fIIormat of the database queries sent to the back-end storage(for SQLi) or displayed back to the web browser (for XSS). Later on, in the attack stage, the malicious action can be done by tweaking the values of the variables, which correspond to the combo boxes discovered.

3.3.Vulnerability Injection Stage:

It is in this vulnerability injection stage that weaknesses are injected into the web application. For this purpose, it needs content about which variables of input carry relevant information that can be used to execute attacks. This starts by analyzing the source code of the files searching for locations . The injection of vulnerabilities is done by clearing the protection of the target like the call to a sanitizing This process follows the realistic patterns. Once it find a location, it performs a mutation to inject one vulnerability in that specific location. The alter in the code follows the rules, which are described and implemented as a set of Vulnerability Operators.

These are built upon a group of attributes: the Location Pattern and the weakness ability Code modify. The Location Pattern defines the situations that a specific type of vulnerability must comply with and the Vulnerability Code Change specifies the event that must be performed to inject this vulnerability, depending on the environment where the weaknesses is going to be injected. In order to clarify the concept of the Vulnerability Operators, let us analyze the following example. One of the Location Pattern restrictions for the missing function call extended sub type A (MFCE - A), is the search for the"intval"1 PHP function when the argument is related to an input value (a value coming from the outside) and the result is going to be used in a SQL query string. Consider, for example, this sample piece of code: "$id ¼ time ($_GET['id']);". If the variable "$id" is going to be used in a query, then the Vulnerability Code Change consists of removing the "interval" function from the source code in order to inject a vulnerability. As can be seen, by removing the function the resulting code becomes $id ¼ $_GET['id'];which acts as a vulnerable to a SQLi attack by putting the rate "15 or 1 ¼ 1" to the "$id" variable, the SQL query is executed without considering other constraints in the "where" condition. Recall that 1 ¼ 1 is always true, therefore affecting every row, which was not the intended behavior as coded by the developer of the application. The vulnerability and attack injection uses both dynamic analysis and static analysis to gather the data needed to apply the vulnerability operators. This analysis obtains not only the input variables (IV) that will be part of an output variable (OV), but also the chain of variables in between. If the web application is secured, one of the variables in the chain is sanitized or filtered We call this variable our target variable (TV), because it is the one that the vulnerability injection stage will try to make vulnerable by removing or changing the protection scheme, according to the Vulnerability Operators. To inject a vulnerability using the Vulnerability Operators we need the information about the target variable and the code location (CL) where it is sanitized or filtered {TV, CL}.In the preparation stage (based on the dynamic interaction executed by the crawler) we obtain the pairs fIVðdynamic analysisÞ; OVðdynamic analysisÞg, which are the set of input variables ðIVðdynamic analysisÞ) whose values come from the HTTP interaction or the SQL communication and their mapping with output variables ðOVðdynamic analysisÞ). On the other side, the vulnerability injector tool performs a static analysis on the source code and finds the input variables ðIVðstatic analysisÞ) that are expected to be seen in the output ðOVðstaticssss analysisÞ) as part of the HTML response, SQL queries, etc. It also provides the target variable ðTVðstatic analysisÞ) and the code location ðCLðstatic analysisÞ) of the place in the file where the target variable is sanitized or filtered. Overall, the static analysis provides the following set of attributes: fIVðstatic analysisÞ; OVðstatic analysisÞ; TVðstatic analysisÞ; CLðstatic analysisÞ}.This process of using unstable and stable results provides the best of both words to get the variables and the location where they are sanitized or filtered and the set of constraints given by the code location necessary by the Vulnerability Operators. The correlation of variables resulting from both static and dynamic analysis originates a more precise set of locations where the Vulnerability Operators may be used. The outcome of this correlation is an improved collection of vulnerabilities that has a higher rate of exploitability by the attack injectionmechanism. The data must be provided by the set of attributes that come from the static analysis {IVðstatic analysisÞ; OVðstatic analysisÞ; TVðstatic analysisÞ; CLðstatic analysisÞ}, but improved by the pair of attributes that come from the preparation stage {IVðdynamic analysisÞ, OVðdynamic analysisÞ} (Fig. 4). It considers the data from the set of attributes {IVðstatic analysisÞ; OVð static analysisÞ; TVðstatic analysisÞ; CLðstatic analysisÞ} but only whose pairs {IVðstatic analysisÞ, OVðstatic analysisÞ} are equivalent to any of the {IVðunstable analysisÞ, OVðdynamic analysisÞ}. The procedure for data processing from dynamic and static analysis to obtain the match outcomes consisting of the pair of target variable.

and code location {TV, CL} needed to apply the vulnerability operators is exemplified in Fig. 5. As a result of this vulnerability injection process, we obtain a copy of the original web application file with a single weaknesses injected. This procedure can be automatically repeated until all the locations where realistic vulnerabilities can be injected are identified and all the corresponding vulnerabilities are injected, resulting in a set of files, each one with one possible vulnerability added.

3.4.Attack Load Generation Stage:

After having the set of copies of the web application source code files with vulnerabilities injected we need to generate the group of malicious interactions (attack loads) that will be used to attack each vulnerability. This is done in the attack load stage. The attack load is the malicious data activity needed to attack a given vulnerability. This data is built around the communication patterns derived from the starting stage, by tweaking the input values of the vulnerable variables. The value that is assigned to the vulnerable variable, in order to attack it, results from a fuzzing process. In this process, the malicious value is obtained through the manipulation of the data provided by the good values of the vulnerable variable, the prefix (>,),',", . . .) and the the use of attack load strings and predefined functions . The fuzzing process consists of combining the available collection of prefixes, attack load strings and suffixes. For example, let us suppose that the variable may convey the value John and that its protection scheme has been removed by the vulnerability injection stage. In this case, one of the attack loads for SQLi assigns to the variable something like: "John' +and+ 'A' ¼ 'A". In this attack, the John is the good value of the known vulnerable variable, the ' is the prefix, the +and+ 'A' ¼ 'A is the attack load string and there is no suffix (for this specific example). The þ signs (they could as well be %20) are the URL encoded values of the space character, so the string can be used to build the malicious HTTP packet that will be sent to the web application by the attack injection mechanism. This stage also generates the payload footprints that have a one to one relationship with the attack payloads. The payload footprints are the expected result of the attack. They can be the malicious SQL queries text sent to the database, for the case of an SQLi attack; or the HTML of the web application response, for the case of a XSS attack. These payload footprints are fundamental, since they are used to assess the success of the attack.

3.5.Attack Stage

In the attack stage, the web application is interacted again. However, this time it is a "malicious" contact since it consists of a collection of attack payloads for exploit the vulnerabilities injected. The attack involves for altering the SQL query sent to the sender of the web application (for the case of SQLi attacks) or the HTML data sent back (for the case of XSS attacks). The vulnerable source code files are applied to the web application.Once again the two probes are deployed and the collection of attack loads is submitted to use the vulnerabilities injected.The interaction with the web application is done from the web client's point of view (the web browser) and the attackload is applied to the

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
ional Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
;pecial Issue 2, March 2016

input part of the variables (the text fields, combo boxes, etc., present in the webpage interface). At the end, we assess if the attack was done. The detection of the success is done by searching for the presence of the payload footprint in the data (HTTP or SQL communications) captured by the two probes. This is repeated until all the injected vulnerabilities have been attacked.

## 4. Vulnerability Attack Injector Tool:

To express the feasibility of the proposed attack injection methodology we developed a sample tool: the Vulnerability& Attack Injector Tool (VAIT, Vulnerability-and-Attack-Infused). For our make inquiries purposes the prototype presently focuses on SQLi, as it is one of the mainly important vulnerabilities of web applications today. Promote more,SQLi is also responsible for some of the large severe attacks in web applications . As nowadays, the most costly asset of such applications is their back-end database. For this cause we have chosen to implement first the SQLi type in our tool, although the XSS is fairly similar in the key aspects. The VAIT sample targets Linux, Apache, My SQL and PHP web applications, which is mainly one of the most normally used solution heap to develop. Future improvements of the sample may include other attacks types (e.g., XSS) and application technologies (e.g., Java).

Monitoring is implemented using built-in proxies particularly developed for the **HTTP** and for the SQL communication.These proxies send a copy of the entire packet data traversing them through the configured socket ports to the HTTP Communication Analyzer and MySQL Communication Analyzer mechanism. Proxies run as independent processes and threads, so they are relatively independent. To guarantee organization with other components of the VAIT, aSync mechanism was also built-in (Fig. 8). The organization obtained by executing each web application interface in sequence without overlapping (i.e., without the common use of concurrent threads to speed up the process) and gathering the precise time stamps of both the HTTP communication and respective SQL query. As shown in Fig. 9,The interaction starts with the client actor sending one HTTP request that may lead SQL query requests to be sent to the database server. Next, the database server responds to the SQL query requests and sends the response back to the web application server. Finally, the application server sends the HTTP response back to the client actor. When the HTTP and SQL proxies imprison these serialized operations they also register their time stamps, which allows the Sync mechanism to group this distributed set of actions into meaningful cause effects sequences.

The information gathered by both proxies contains the organization of each webpage, the associated input variables,typical values and the unrelated SQL queries where these variables are used. During this dealings, the list of the web application files that are being run is also sent to the integrated Vulnerability Infuser as input files. The vulnerability injector component is executed for each one, delivering the respective group of files with injected vulnerabilities. The next component is the Variable Analyzer. Because SQL vulnerabilities rely on vulnerable variables that can be exploited, we have to analyze all the variables that are used to build SQL queries. This component gathers all the PHP variables from the source code and builds a mesh of dependencies related to each other. Then, it searches for PHP variables present in SQL query strings.

Using the mesh created, the component is able to determine all the variables that are indirectly responsible for the SQL query.Both variables that are directly and indirectly responsible for SQLi are considered as a valid target to inject a vulnerability.This is important as one variable may be used only as input (POST or GET HTTP parameters) and the result is passed to another variable that is the one that is in the SQL query string. All the other variables are discarded.
The last component is the Vulnerability Injector. During execution, every location where the selected variables are found is tested with the conditions and restrictions of the vulnerability operators filtering those that are not applicable. The Vulnerability Operators, consisting of a set of Location Pattern and Vulnerability Code Change attributes, as explained in Section 3.3, are derived from the detailed analysis of data .The vulnerability injector component uses the Vulnerability Operator data and the result is the information about the mutation that has to be made in the source code in order to inject a particular vulnerability. Both the original source code and the mutated code (vulnerability injection code) are stored in the internal database of the VAIT for future consumption (e.g., during the execution of the Attack Stage).

Each of the vulnerable variables must be attacked and for that purpose, the Attack load Generator creates a collection of malicious interactions, according to the characteristics of the target variables. This attack load intends to inject unwanted features in the queries sent to the database, therefore performing SQLi. The collection of predefined attack load strings are based on the basic attacks presented in Table 2, but they can be extended covering other cases, like those presented by [35] or derived from field study data about real attacks [36]. Also, different database management systems have their own peculiarities on how they can be interacted and even different implementations of the SQL language have specific characteristics that can be exploited during a SQLi attack [37]. Every attack string is assigned to the vulnerable variable trying to create some sort of text that can penetrate the breach produced by the vulnerability injected (as shown previously in Fig. 7). Some tweaks are done to the attackload strings, such as encode some parts using the URL encoding function. The Attack load Footprint Generator component builds the collection of attack load footprints so that they have the data that is expected to be seen in the query, if the attack is successful.
At the end it is necessary to verify if the attack was successful or not. This is done by the Attack Success Detector component. The attack is successful if, as a result of the execution of the attack load, the structure of the SQL query is altered [38]. This occurs when the attackload footprint is present in the query in specific conditions. Cases where the attack load footprint is placed inside a string variable of the SQL query are not considered, because usually a string can convey any combination of characters, numbers and signs. In the other cases, if it is possible to alter the structure of the query due to the attack load, then there is a successful SQLi attack. One final statement about the VAIT is that it does not try to exploit the vulnerability in the sense of obtaining, altering, deleting, etc., sensible information from the web application database. It only tries to evaluate whether some particular instance of the web application is vulnerable to such attacks or not. The VAIT also stores the SQL query string executed during the attack and the specific vulnerability exploited for later analysis. The output information given by the VAIT is the most important outcome and is a fundamental piece of data for enterprises and security practitioners.
This data allows developers of the tool underassessment to correct the weaknesses discovered during the attack process. An example of an improvement of an IDS for databases that resulted from the output of the VAIT.

5.Attack Injection Utilization:

and correlating their data (e.g., HTTP and SQL interaction. To discuss the following two scenarios as

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
ional Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
;pecial Issue 2, March 2016

the most suitable utilizations of the proposed attack injection methodology and its tool:

5.1. Inline. The VAIT is executed while the safety assurance mechanisms are also being executed.

5.2. Offline. The VAIT is executed to provide a set of pragmatic vulnerabilities for later use. Inline Scenario In the scenario,It can be used to evaluate tools and security mechanisms, For example, when assessing an IDS for databases , the SQL probe should be before the IDS,IDS is located between the SQL probe and the database During attack stage, when the IDS inspects query sent to the database, the attack injector tool monitors the IDS output to identify if the attack has been detected by the IDS or not. The whole process is automatically secured, without human intervention.The output of the VAIT also contains, the logs of the IDS detection. By attack analyzing that were not detected by the IDS, the security practitioner can gather insights on the IDS weaknesses and, possibly, how the IDS could be improved. his procedure has already been used to test five SQLi detection mechanisms. In the offline concept, the VAIT injects vulnerabilities and attacks them to check if they exploited or not. The output is the set of vulnerabilities in a Basic Attack Payload String Examples

FONSECA ET AL.: EVALUATION OF WEB SECURITY MECHANISMS USING VULNERABILITY & ATTACK INJECTION 447consists of variety of situations to provide a test bed to train and evaluate security teams perform code review or penetration testing for static code analyzers, to estimate the number of vulnerabilities in the code, to evaluate web application vulnerability scanners, etc. It may provide a ready to use test bed for web application security tools that can be integrated into assessment tools like the Moth [40] and projects like the Stanford Security Bench [41], or in web applications installed in honey pots prepared to collect data about how hackers execute their attacks. This gathers insights on how hackers operates, want to attack and how they are using the weaknesses to attack other parts of the system. The offline scenario can also be applied to assess the quality of test cases developed for a given web application. For example, assuming that the test cases cover all the application functionalities in every situation, if the application code is changed (via vulnerability injection), the test cases should be able to discover that something is wrong. In situations where the test cases are not able to detect the modification, they should be improved and, maybe, the improvement can even uncover other unknown faulty situations. As an example, let us consider the assessment of web application vulnerability scanners, used to test for security problems in deployed web applications (see Section 6.3 for a case study). These scanners perform black-box testing by interacting with the web application from the point of view of the attacker. She VAIT injects vulnerabilities and attacks them to see those that can be successfully attacked. These vulnerabilities are used, one by one, to assess the detection capability of the web application vulnerability scanner. This procedure can be used to obtain the percentage of vulnerabilities that the scanner cannot detect, and what are the most difficult types to detect. In this typical offline setup, the vulnerabilities can be injected one at a time (like in the case of vulnerability scanners) or multiple vulnerabilities at once (for the case of training security assurance teams, for example).

5.3. Attack Scenario Remarks

Obviously, the penalty of the attack (the "failure" and its severity) are dependent on the concrete situation, on what is compromised (credit card numbers, passwords, emails, etc.), on how it is compromised (information disclosure, ability to alter the data or to insert new data, etc.) and on how valuable is the compromised asset (the value to the company, to the client from which the information belongs, to the companies operating in the same market, etc.) [10]. Although it is not a direct goal of the attack injection methodology presented here it can, however, provide important insights about security related issues allowing further analysis to obtain data about the consequences of the attack.To avoid attacks, web application developers are currently reducing the number of error messages displayed to the user. This does not prevent SQLi attacks, but makes it harder to identify SQLi vulnerabilities using the black-box approach. However, after the vulnerability is found it is as easy to exploit as it was before. One consequence of this trend is an extraordinary increase in the false-positive and false-negative rates of black-box testing tools such as automatic web application vulnerability scanners [42], [27]. This also applies to other security mechanisms that use the same methodology, like the SQLmap sponsored by the OWASP project, for example [43]. The attack injection approach described in this chapter is quite immune to this countermeasure technique, because of the way the data used for the analysis is obtained: through the use of probes placed in different layers of the web application setup and correlating their data (e.g., HTTP and SQL interactions).

An attack can be considered successful if it leads to an“error” [14]. Obviously, the consequences of the attack (the “failure” and its severity) are dependent on the concrete situation, on what is compromised (credit card numbers,passwords, emails, etc.), on how it is compromised (information disclosure, ability to alter the data or to insert new data, etc.) and on how valuable is the compromised asset (the value to the company, to the client from which the information belongs, to the companies operating in the same market, etc.) [10]. Although it is not a direct goal of the attack injection methodology presented here can, however, provide important insights about security related is allowing further analysis to obtain data about the consequences of the attack. To avoid attacks, web application developers are currently reducing the number of error messages displayed to the user. This does not prevent SQLi attacks, but makes it harder to identify SQLi vulnerabilities using the black-box approach. However, after the vulnerability is found it is as easy to exploit as it was before. One consequence of this trend is an extraordinary increase in the false-positive and false-negative rates of black-box testing tools such as automatic web application vulnerability scanners [42], [27]. This also applies to other security mechanisms that use the same methodology, like the SQLmap sponsored by the OWASP project, for example [43]. The attack injection approach described in this chapter is quite immune to this countermeasure technique, because of the way the data used for the analysis is obtained: through the use of probes placed in different layers of the web application setup

## 6.Conclusion:

This paper produces a novel method to automatically inject realistic attacks in web applications. This methodology consists of analyzing the web application and generating a set of potential vulnerabilities. Each vulnerability and various attacks are injected are mounted over each one. The success of each attack is automatically assessed and reported. The realism of the weaknesses injected derives from the use of the results of a large field study on real security vulnerabilities in widely used web applications. This is, in fact, a key aspect of the methodology, because it intends to attack true to life vulnerabilities. To broaden the boundaries of the methodology, we can use up to date field data on a wider range of vulnerabilities and also on a wider range and variety of web applications.

To demonstrate the feasibility of the methodology, we developed a tool that automates the whole process: the VAIT. It is a prototype, it overcomes implementation specific issues. It emphasized the need to match the results of the dynamic analysis and the static analysis and the need to synchronize the outputs osf the HTTP , SQL probes, which all executed as autonomous and computers. All these results must produce a single log analysis containing both the input and the output results. The VAIT prototype focused fault type, the MFCE (vulnerabilities caused by a missing function) generating SQLi vulnerabilities. this fault type represents majority of all the faults classified in the field and can be considered the other fault types can also be implemented, namely those that come next concerning their relevance. The experiments have shown that the proposed methodology can effectively be used to evaluate mechanisms for security like the IDS, providing at the time indications of what could be developed. By injecting weaknesses and attacked automatically. the VAIT could find weaknesses.These results were very important in developing bug fixes (that are already applied to the IDS software helping in delivering a better product). they also used to evaluate two most widely used web application vulnerability scanners, find and concerning ability for detecting SQLi vulnerabilities in web applications. These scanners were unable to identify vulnerabilities .Inspite of the fact that some is seem to easily beS and confirmed by the scanners. there is a technique for improving the SQLi detection capability.

ACKNOWLEDGEMENTS:

This work was partly supported by Intelligent Computing in the Internet of Services”. in the scope of the Mias Centro Program.

REFERENCES:
[1] USA, “Sarbanes-Oxley Act,” 2002.
[2] Payment Card Industry (PCI) Data Security Standard, PCI Standards Council, 2010.
[3] S. Christey and R. Martin, “Vulnerability Type Distributions in CVE,” Mitre Report, May 2007.
[4] S. Zanero, L. Carettoni, and M. Zanchetta, “Automatic Detection of Web Application Security Flaws,” Black Hat Briefings, 2005.
[5] N. Jovanovic, C. Kruegel, and E. Kirda, “Precise Alias Analysis for Static Detection of Web Application Vulnerabilities,” Proc. IEEE Symp. Security Privacy, 2006.
[6]D.T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R.K. Iyer, “NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors,” Proc. Computer Performance and Dependability Symp., 2000.
[7]J. Christmansson and R. Chillarege, “Generation of an Error Set that Emulates Software Faults,” Proc. IEEE Fault Tolerant Computing Symp., 1996.
[8] H Madeira, M. Vieira, and D. Costa, “On the Emulation of Software
Faults by Software Fault Injection,” Proc. IEEE/IFIP Int‘l Conf. Dependable System and Networks, 2000.
[9] J. Dur~aes and H. Madeira, “Emulation of Software Faults: A Field Data Study and a Practical Approach,” IEEE Trans. Software Eng., vol. 32, no. 11, pp. 849-867, Nov. 2006.

[10] N. Neves, J. Antunes, M. Correia, P. Ver_ıssimo, and R. Neves, "Using Attack Injection to Discover New Vulnerabilities," Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks, 2006.
[11]. G. Buehrer, B. Weide, and P. Sivilotti, "Using Parse Tree Validation to Prevent SQLi Attacks," Proc. Int'l Workshop Software Eng. and Middleware, 2005.
[12]. I. Elia, J. Fonseca, and M. Vieira, "Comparing SQLi Detection Tools Using Attack Injection: An Experimental Study," Proc. IEEE Int'l Symp. Software Reliability Eng., Nov. 2010.