International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-2 Issue-5, April 2013

R-WASP: Real Time-Web Application SQL Injection Detector and Preventer

Munqath H. Alattar S.P. Medhane

Abstract- In the real time word, there are many online systems those are major part of software systems in order to make them publically available to perform the remote operations. These online systems are vulnerable to different types of web based attacks. Here in this project we are considering the one such web based attack and its prevention technique in real time web applications as well as presenting the ways to implement same approach for binary applications. Previously, the approach called WASP was proposed as efficient web application SQL injection preventer using the datasets. However, this tool was not evaluated over real time web applications; we did not get its accuracy for prevention of real time web application SQL injection attacks, even though it's having high accuracy during its tested results over datasets. Therefore, in this research work we are extending the WASP approach to real time environment in order to evaluate its effectiveness as well as to collect a valuable set of real legal accesses and, possibly, attacks. In addition to this, we are presenting the same approach for binary applications. This new approach or tool we called as R-WASP.

Index Terms— WASP, SQL injection attack, Binary applications.

I. INTRODUCTION

As in introduction, here we will discuss the approaches those are used in WASP. To solve these issues the subsequent approaches are utilized in WASP own designed and developed framework. Positive tainting differs from ancient tainting (hereafter, negative tainting) as a result of its supported the identification, marking, and pursuit of sure, instead of un trusted data. This abstract distinction has important implications for the effectiveness of our approach; therein it helps address issues caused by wholeness within the identification of relevant knowledge to be marked. wholeness, that is one in all the main challenges once implementing a security technique supported dynamic tainting, has terribly totally different consequences in negative and positive tainting. Within the case of negative tainting, wholeness results in trusting knowledge that ought to not be sure and, ultimately, to false negatives. Wholeness could therefore leave the applying at risk of attacks and may be terribly tough to observe, even when attacks really occur, as a result of they will go utterly unmarked. With positive tainting, wholeness could result in false positives; however it'd ne'er lead to associate degree SQLIA escaping detection. Moreover, as explained within the following, the false positives generated by our approach, if any, square measure seemingly to be detected and simply eliminated early throughout prerelease testing.

Manuscript published on 30 April 2013.
*****Correspondence Author(s)
Munqath H. Alattar, Manuscript received April, 2013.Munqath H. Alattar Information Technology Department, College of Engineering, BharatiVidyapeeth University, Pune, India
S.P. Medhane, Prof.S.P. Medhane, Information Technology Department, College of Engineering, BharatiVidyapeeth University, Pune, India.

Positive tainting uses a white-list, instead of a black-list, policy and follows the final principle of fail-safe defaults, as printed by Saltzer and Schroeder: just in case of wholeness, positive tainting fails during an approach that maintains the safety of the system [1].

Within the context of preventing SQLIAs, the abstract benefits of positive tainting square measure particularly important [1]. The approach during which net applications produce SQL commands makes the identification of all un trusted knowledge particularly problematic and, most significantly, the identification of most sure knowledge comparatively easy. This approach inside the WASP proves the effectiveness of WASP tool for preventing the 1200 different kinds of SQL Injection attacks. In this project we are presenting the real time framework for WASP called AS R-WASP, which also be implemented for binary applications completely different technologies and recommending approaches for higher results can invariably exist and can be the explanation of latest researches.

II. PROBLEM DEFINITION

As we study in WASP [1] paper, WASP tool presented the novel highly automated approach for protecting Web applications from SQLIAs. WASP approach consists of: 1) Identifying trusted data sources and marking data coming from these sources as trusted, 2) Using dynamic tainting to track trusted data at runtime, and 3) Allowing only trusted data to form the semantically relevant parts of queries such as SQL keywords and operators. In addition to this, WASP eliminating the problem of false negatives those are resulted from the incomplete identification of all un trusted data sources. We observed in [1], WASP shows the great level of effectiveness over the existing methods during its evaluation using datasets at simulation level.

However, we found the legitimate problem of effectiveness of WASP under real time settings. Any security tool does not becomes effective until and unless its effectiveness is evaluated using the real time environment. WASP tool was not tested using already deployed web applications was not tested to detect the real time attacks.

III. MOTIVATION

The problem in the existing system is that the SQL attacks are not detected in a proper format as well as in real settings. There are various methods used in the detection and prevention of SQL injection attacks but the prevention methods do not use a standard format to prevent the attacks. In the existing system the tools that are used to detect the SQL injection attacks are using various another algorithms or methods to find the injections that are performed, however those are not yet tested under the real time settings. This gives us immense motivation to extend this approach towards already deployed web applications.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

IV. EXISTING WORK AND LIMITATIONS

There are methods used in order to prevent SQL attacks, and one of them is the use of Proxy Filters, which is a system of enforcing input validation rules on data that are flowing the to a web application. The developers offer constraints through the Security Policy Descriptor Language (SPDL), thus specifying the transformations that are applied for application of parameters that flowing Web page to the application server (Boyd and Keromytis,292). This method also allows developers to express their policies since SPDL is highly expressive, though the approach is human-based and defensive programming, thus requiring the developers to identify the data that require filtering.

Figure 1: Depicts the filtering which occurs between the server and the Internet

The diagram (Fig.1) depicts the filtering which occurs between the server and the Internet in order to prevent an injection by an attacker using the clients to the web servers with SQL database.

The other preventive method relates to the use of Combined Static and Dynamic Analysis, through a model referred to as AMNESIA, which is a technique integrating static analysis and monitoring runtime. AMNESIA applies statistical analysis that develops models of different forms of queries that are generated by an application at a point of access to the database (Halfond and Orso, 174). In fact, this model intercepts queries sent to the database, and checks query against the model that is built statically, thus providing a basis for identifying the queries that violate the model, hence preventing them from executing on the database, though this model has a constraint associated with dependence on the precision of the static analysis for developing the models.

The other preventive method is the New Query Development Paradigms, which entails two resent approaches: SQL DOM and Safe Query Objects, and application of encapsulation of database queries that offer a safe and reliable way of accessing the database. This method provide an effective way of avoiding SQL attacks, by altering the process of building the query from an unregulated process that utilizes strings concatenation to a process involving a type check of API (McClure and Krüger, 88). Therefore, this method allows a systematic application of best coding practice like filtering of input and checking of user input, thus altering the development of paradigm that create SQL queries can eliminate the coding practice that facilities vulnerabilities SQLIAs.

Nevertheless, this method has a drawback associated with the requirement of a developer to learn and apply new programming paradigm or query development process. Consequently, focusing on the use of a new development process, there is no provision of any form of protection for a legacy system.

Scope of Research:

Application-level vulnerabilities, that are believed to account for 70% to 90% of overall flaws, are currently the most focus of attackers and researchers as compared to simulation primarily based approaches planned to stop the SQL injection attacks. On-line applications (websites and services) are particularly in danger thanks to their universal exposure and their intensive use of the firewall-friendly hypertext transfer protocol. Moreover, information security is simply too usually unnoticed in favor of net and application server security, leading to backend databases being a serious target for attackers that are ready to use them as straightforward entry points to organizations' networks.

Protecting online applications (e.g. websites) and web services against SQL Injection Attacks has thus become a major concern for organizations, which face threats that can go far beyond the expected reach of the public web or application server. While several effective prevention methods have been developed, ensuring full protection against SQL Injections remains an issue on a practical level.

V. SYSTEM ANALYSIS

The first goal of the proposed system was to develop as well as implement a highly automated technique against SQLIAs that is able to detect, stop, and report injection attacks before they reach the database and do any harm.

Figure 2: Overview of Proposed Approach

Figure 2 provides a general, intuitive overview of the proposed approach. Given a previously developed Web application, our tool would automatically transform the application into a semantically equivalent application that is protected from SQLIAs. The second goal of the project was to develop a testbed that could be used by us and by other researchers and practitioners to evaluate tools for SQL injection detection and prevention. Third goal of the project is to prepare the set of online websites on which we are going to test our tool. The fourth goal of the project was to make the tools developed during the project industrial strength and to commercialize them; in short we have to evaluate that tool under the real time environment. Hence this new tool is called as R-WASP (Real-WASP).

System Description

The main approach of system is to implement a tool which detect and prevent SQLIAs based on a technique, called R-WASP. Here we have to implement proposed tool as a stand-alone tool and empirically assessed its performance on a set of real applications and attacks as well as binary applications. We have to first implement the algorithms of R-WASP using .net infrastructure, and then prepare the set of real time applications on which we are testing our algorithms effectiveness.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

VI. PROPOSED ALGORITHM

In this paper we are focusing to use the WASP approach as proposed approach by considering the real time set of online applications and binary applications as input. Proposed modified architecture is depicted in following figure 3. Also below points explaining the methods we used from WASP in newly proposed *R-WASP*.

Still to date, most of the tools developed for protecting Web Application from SQL Injection attack use/followed Traditional Dynamic Tainting. Traditional Dynamic Tainting identified un trusted String as tainted, track the flow of tainted data at runtime, and prevents this data from being used in potentially disaster ways. To make changes in previous approach and develop a new improved tool which followed Dynamic Tainting.

Unlike existing dynamic tainting techniques, our approach is based on the novel concept of positive tainting, that is, the identification and marking of trusted for both real time and binary applications. Second, our approach performs accurate and efficient taint propagation by precisely tracking trust markings at the character level. Third, it performs syntax-aware evaluation of query strings before they are sent to the database and blocks all queries whose non literal parts (that is, SQL keywords and operators) contain one or more characters without trust markings.

4.1 Real Time Based Positive Tainting

As we studied in [1], positive Tainting is based on identification of the trusted data rather than untrusted data. Traditional Tainting (negative tainting) follows the identification of untrusted data and here positive and negative tainting differs. This conceptual difference has significant implications for the effectiveness of our approach, in that it helps address problems caused by incompleteness in the identification of relevant data to be marked. Incompleteness leaves the Web Application vulnerable to SQL injection attacks. In negative tainting detection of attacks is very difficult. Hence we use positive tainting in our approach. Identifying trusted data in Web Application is often straight forward and always less error prone. Here positive tainting will directly sense the real time traffic from set of input web applications.

4.2 Accurate as well as Efficient Taint Propagation

Taint Propagation is carried at runtime. It consists of identifying taint markings associated with data, while the data is used and manipulated by users at runtime. Taint Propagation needs to be carried out accurately otherwise it would cause the data to be misused. Our approach consists of:

1) Identifying taint markings at correct level of granularity
2) Precisely accounting for the affect of functions that operate on the tainted data.

The data consists of characters. Hence to achieve accuracy tainting at character level is carried in our approach. Here Strings are constantly broken into substrings for building SQL quires.

4.3 Syntax Aware Evaluation

Positive tainting helps us to create taint markings during execution but for achieving more security we must be able to use the taint markings to distinguish legitimate from malicious queries.

The key feature of Syntax aware evaluation is that it considers the context in which trusted and untrusted data is to make sure that all parts of query other than string or numerical ,literals consists only of trusted characters. Before the String is sent to the database for execution syntax aware evaluation of a query string is performed. We use SQL parser for creating tokens of the String. The tokens correspond to keywords, operators & literals. The technique that iterates through the tokens & checks whether the tokens other than literals contain only trusted data. If all tokens are identified as trusted the query is identified as safe and passed further, if an attack is detected a developer specified action can be invoked. The developers provide with the external data sources which should be trusted and our technique would mark and treat data coming from these sources accordingly.

Following is the framework for ***R-WASP:***

Online set of applications

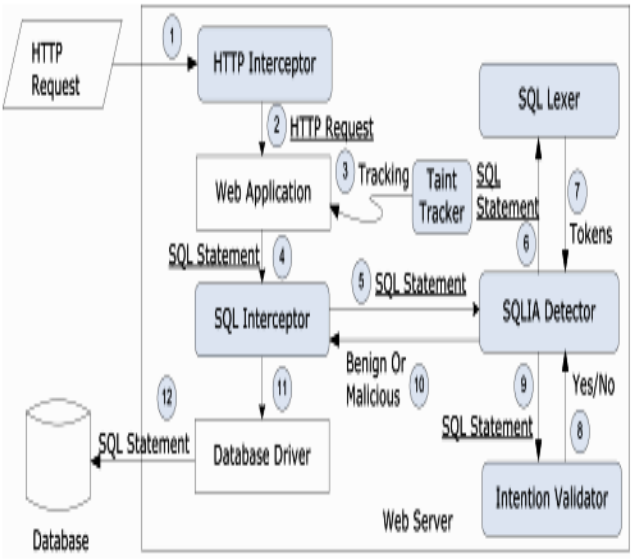

Figure 3: Proposed RWASP Framework

VII. EXPERIMENTAL ANALYSIS

Requirement Analysis

Since ASP.NET has no open mechanism for intercepting SQL statements, we used .NET pro ling API [Pie01] and Microsoft Intermediate Language (MSIL) re-writing techniques [Mik03] to intercept SQL statements. Microsoft .NET profiling API is a standard mechanism for application or tool developers to instrument the behavior of a .NET program by providing components that register events of interest, such as just-in-time compilation or method invocation. Right before a .NET managed method is invoked, Common Language Runtime (CLR) loads the .NET assembly that contains the method code into memory, and compiles it into executable by the target CPU (a process called just-in-time compilation). ASP.NET version of SQL Interceptor provides components that register just-in-time compilation events of database-access classes such as SQLCommand and OleDbCommand. In the event handler,SQL Interceptor employs MSIL re-write technique to install a hook that calls SQLIA Detector module in the beginning of each method of interest.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Those mechanisms are supported by all versions of .NET, and can be deployed into existing ASP.NET web application dynamically without access to application source code. ASP, executed in the domain of COM (Common Object Model) [Mic08c], has no default mechanism for intercepting SQL statements, either.

Hardware and Software requirements

HARDWARE REQUIREMENTS

- Windows XP
- RAM – 1GB
- Hard Disk - 40GB

SOFTWARE REQUIREMENTS

- .Net Framework
- IIS 7.0
- MySQL
- SQL Server

Applications

- Most of the banking operations based on online communications.
- Online Shopping Web portals.
- Social Networking
- Web Surfing

Results

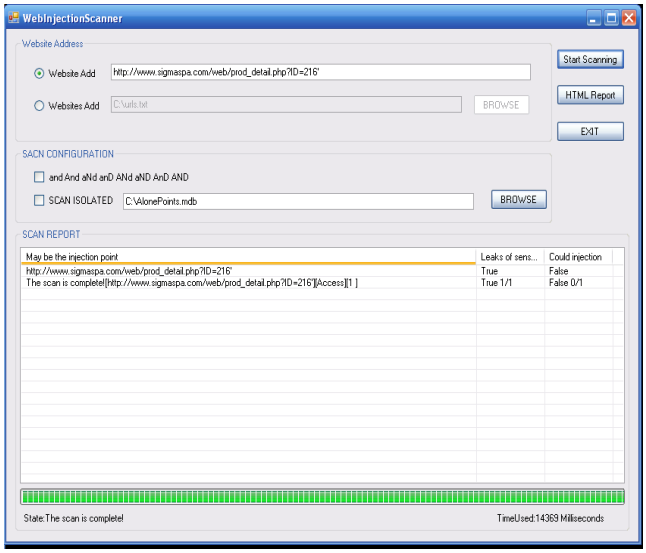

Figure 4: R-WASP Framework

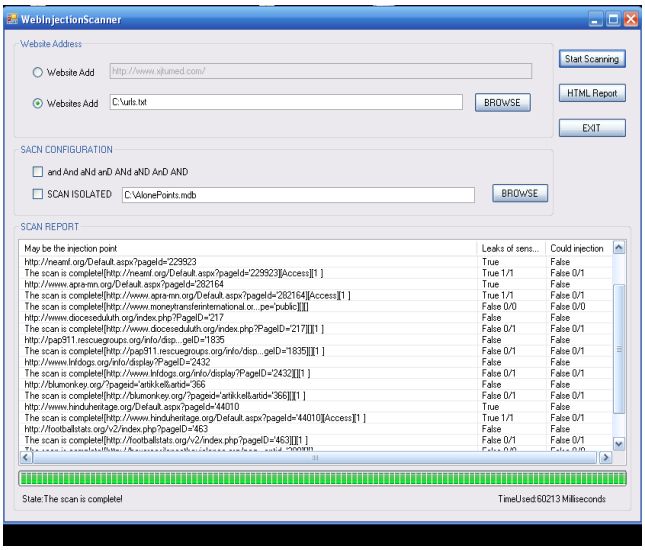

Figure 5: R-WASP Framework for detection and prevention of SQL injection from given website

VIII. CONCLUSION AND FUTURE WORK

SQL injection vulnerabilities are omnipresent and dangerous; nevertheless several web applications deployed nowadays are still susceptible to SQLIAs. Though recent analysis on SQLIA detection and prevention has with success addressed the shortcomings of existing SQLIA countermeasures, the effort required from web developers like application source code analysis/modification or modification of the runtime environment results in restricted adoption of those countermeasures in real world. During this paper, we've given a replacement approach to the planning of protection mechanisms for existing web applications known as R-WASP. R-WASP relies on the core ideas of WASP. The approach permits treatment of net applications as black boxes for the aim of runtime detection and hindrance of SQLIAs. This work additionally contributes with a style, implementation, and analysis of the planned approach within the kinds of R-WASP. Our expertise and analysis of R-WASP indicate that it's effective, efficient, moveable among back-end databases, simple to deploy while not the involvement of net developers, and doesn't need access to the appliance source code.

REFERENCE

1. "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation" , William G.J. Halfond, Alessandro Orso, Member, IEEE Computer Society, and Panagiotis Manolios, Member, IEEE Computer Society, 2008.
2. Boyd, Stephen, and Keromytis, Angelos. "SQLrand: Preventing SQL injection attacks". In Proc. of the 2nd Applied Cryptography and Network Security. Conf. (ACNS 2004), pages 292–302, Jun. 2004.
3. Gould, Carl, Su, Zhendong and Devanbu Premkumar. "Static Checking of Dynamically Generated Queries in Database Applications". In Proc. of the 26th Intern. Conf. on Software Engineering (ICSE 2004), pages 645–654, May 2004..
4. Haldar, Vivek, Chandra, Deepak and Franz, Michael. "Dynamic taint propagation for java". In Proc. of the 21st Annual Computer Security Applications. Conf. (ACSAC 2005), pages 303–311, Dec. 2005.
5. Halfond, William and Orso Alessandro. "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks". In Proc. of the IEEE and ACM Intern. Conf. on Automated Software Engineering (ASE 2005), pages 174–183, Nov. 2005.
6. Andrews, M.: Guest Editor's Introduction: The State of Web Security. IEEE Security and Privacy, 4, 4, 14--15 (2006)
7. Janot, E.: SQLDOM4J: Preventing SQL Injections in Object-Oriented Applications. Master thesis, Concordia University College of Alberta (2008), http://waziboo.com/thesis
8. McClure, R., Krüger, I.: SQL DOM: Compile Time Checking of Dynamic SQL Statements. In: 27th IEEE International Conference on Software Engineering, pp. 88--96. IEEE Press, New York (2005)
9. Halfond, W., Orso, A.: Preventing SQL Injection Attacks Using AMNESIA. In: Di Nitto, E., Murphy, A.L. (eds.) 28th ACM/IEEE International Conference on Software Engineering, pp. 795--798. ACM, New York (2006)
10. Yuhanna, N.: The Forrester Wave™: Enterprise Database Auditing And Real-Time Protection, Q4 2007.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)