Journal of Information Security and Applications 40 (2018) 199–216

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel

Paul R. McWhirter, Kashif Kifayat*, Qi Shi, Bob Askwith

Department of Computer Science, Liverpool John Moores University, Liverpool L3 3AF, UK

ARTICLE INFO

Article history:

Index Terms:
Intrusion detection
SQL Injection Attacks
Data mining
String Subsequence Kernel
Support vector machine
Supervised learning

ABSTRACT

SQL Injection Attacks are one of the most common methods behind data security breaches. Previous research has attempted to produce viable detection solutions in order to filter SQL Injection Attacks from regular queries. Unfortunately it has proven to be a challenging problem with many solutions suffering from disadvantages such as being unable to process in real time as a preventative solution, a lack of adaptability to differing types of attack and the requirement for access to difficult-to-obtain information about the source application. This paper presents a novel solution of classifying SQL queries purely on the features of the initial query string. A Gap-Weighted String Subsequence Kernel algorithm is implemented to identify subsequences of shared characters between query strings for the output of a similarity metric. Finally a Support Vector Machine is trained on the similarity metrics between known query strings which are then used to classify unknown test queries. By gathering all feature data from the query strings, additional information from the source application is not required. The probabilistic nature of the learned models allows the solution to adapt to new threats whilst in operation. The proposed solution is evaluated using a number of test datasets derived from the Amnesia testbed datasets. The demonstration software achieved 97.07% accuracy for Select type queries and 92.48% accuracy for Insert type queries. This limited success rate is due to unsanitized quotation marks within legitimate inputs confusing the feature extraction. Using a test dataset that denies legitimate queries the use of unsanitized quotation marks, the Select and Insert query accuracy rose.

1. Introduction

SQL Injection Attacks (SQLIAs) involve the crafting of user inputs in order to perform actions beyond the intended function of a web application [1]. By the identification of the input fields associated with the dynamic generation of queries ([23]; Tajpour et al., 2012), the adversary can probe the database data values, the layout of the database (known as the database Schema), perform remote procedures and escalate their privilege on the Database Management System ([2]; Balzarotti et al., 2008). Databases often contain significant quantities of confidential information. As a result it can prove to be lucrative for malicious users of web applications to create queries to resolve data they are not authorized to view or alter. SQL Injections are one of the most serious threats to web applications. It is ranked number one in the Open Web Application Security Project (OWASP) Top Ten Application Security Risks in 2013 [3]. This is due to as many as 98% of web applications having at least one security vulnerability resulting in an increase in SQL injection attacks by ten percent [4].

Our solution to the SQLIA problem is the implementation of Machine Learning methods capable of detecting malicious queries based on information from the structure of the query strings learned from a training set of queries produced during runtime. This structural information is extracted using a Gap Weighted String Subsequence Kernel (GWSSK) function [5]. This function computes the similarity of unknown query strings to preselected training query strings. A Support Vector Machine (SVM) classifier uses these similarity measurements to determine if the unknown query is normal or malicious by determining a decision boundary which maximizes the distance between the two classes [6]. Our method is a form of black box method [2].

This method does not require the re-engineering of SQL-dependent web applications or the full disclosure of their source code. This is a flaw of many previous methods [2]. There are also some solutions that are easily circumvented by attackers constructing novel attacks [19]. As our method uses a probabilistic classifier in the form of the SVM classifier, unknown queries with query structures which deviate from the training dataset are still likely to be determined as malicious due to the extracted simi-

* Corresponding author.
E-mail addresses: P.R.McWhirter@2014.ljmu.ac.uk (P.R. McWhirter), K.Kifayat@ljmu.ac.uk (K. Kifayat), Q.Shi@ljmu.ac.uk (Q. Shi), R.J.Askwith@ljmu.ac.uk (B. Askwith).

https://doi.org/10.1016/j.jisa.2018.04.001
2214-2126/© 2018 Elsevier Ltd. All rights reserved.

larity information. Our solution does have two clear limitations. Our method must be placed between the web application and the database. This introduces hardware overhead required to operate the detection and prevention solution [20,22,24]. Additionally, the detection algorithms are never going to have perfect detection accuracy and therefore issues related to false negatives which can inflict database damage and false positives that can prevent normal operation of a database must be mitigated [21].

Our key contribution is the demonstration of the viability of the GWSSK and SVM algorithms in the high-performance classification of SQL query strings during real-time operation of a database application. This is shown through classification accuracy and time complexity experiments on a dataset of SQL queries exhibiting a wide-range of normal and malicious features. The novel GWSSK method in the automatic extraction of informative features of SQL queries allows for the elimination of biases produced by manually created features potentially improving the accuracy of the SQLIA classification task.

The rest of this paper is structured as follows. In Section 2, the descriptions of related works are presented. In Section 3, the framework of the proposed solution is discussed and the contribution of this paper is clarified. In Section 4, the feature extraction at the core of this solution is defined as the main contribution of this research. In Section 5, the experimental results of the demonstration software for the proposed method are evaluated. These results are then discussed in chapter 6. The final conclusions and proposals of future work are provided within chapter 7.

2. Related works

Research into securing web applications from SQL Injection Attacks has proposed two differing approaches [2]. The first approach involves the rewriting of application source code within the web application and possibly, stored procedures within the database to conduct sufficient input validation. The correct application of these techniques can render a web application secure to injection commands but it comes with a major disadvantage. Completed web applications require redevelopment to incorporate the defensive procedures. However, this is the best way to protect a system from attacks if the system is currently in development and not yet complete. The costs associated with the changing of software vastly increase later into the development cycle.

NoTamper is a black-box testing method designed to determine vulnerabilities in the server-side code. This allows vulnerabilities to be patched although with a severe cost if vulnerabilities are not detected [16]. AMNESIA is another vulnerability exploration method that combines a static analysis of the web application code with runtime monitoring [15]. SQLGuard was proposed as method of analyzing query parse trees both before and after user-input inclusion. This allows the execution of the user-input to be explored [9]. CANDID is another source code analysis method that retrofits the source code with additional candidate queries. The runtime queries can then be compared to these to determine any illegal executions [17].

The second approach involves the deployment of additional software designed to screen the queries generated by a web application before their execution on the database. These software solutions utilize a wide range of techniques and are often significantly less expensive to deploy into an active system. Unfortunately, they often suffer from the disadvantage of not being a complete solution to the problem. Many solutions are unable to detect every type of SQL Injection Attack leaving an avenue for attackers to exploit. They can also be prone to false positive and false negative events where the detection algorithms identify legitimate queries as malicious and block them or allowing malicious queries through resulting in a security breach.

SQLProb is a proxy-based architecture to prevent SQL Injection Attacks [7]. The solution defines a list of queries produced by a web application. It processes all possible queries produced by the typical operation of the web application. These queries are then collected by the proxy software to produce a sample set of SQL queries from the web application. The proxy filter then detects inbound queries and intercepts them. An enhanced Needleman–Wunsch algorithm [25] originally designed for the alignment of string-based genetic data is used to determine the user input within the full query string. The algorithm determines what substring(s) within the query string to remove to gain the closest comparison to the sample queries. This removed data is the input string(s) within the query string. Upon the determination of the user input, the query string is then used to generate a parse tree. A depth-first-search is then conducted to identify the leaf nodes. If a non-leaf node is discovered that has descendent leaf nodes that are only sourced from the user input then the query string that generated the parse tree is malicious. The malicious queries are then rejected by the proxy software leaving only normal queries to be relayed to the database.

A novel method using the Data-Mining of database logs was proposed to detect SQLIAs [8]. The database log files were used to identify queries executing on the database. This file contains information on the query string and the operations performed by the query execution. The solution first generates a query tree [9]. These query trees were used to generate feature vectors using feature extraction. A set of rules defined by the solution developers transform the string and numerical data from the query tree into a multidimensional numerical vector array. A training dataset of these feature vectors containing samples of normal and malicious queries was used to train a SVM to generate a decision rule for the testing of future queries. Kernel functions were then used to allow the solution to determine a non-linear decision rule. Newly logged queries are transformed into query trees from their associated log, composed into feature vectors and compared by the SVM to the decision rule obtained during the training phase. This solution produced very high accuracy of 99.9% for select and insert queries and 99.6% for stored procedures. The primary disadvantage is that this solution can only be used for attack detection and not prevention. This is due to the simple fact that the query logs that the testing criteria are determined from are only produced when a query is executed.

The combination of static and dynamic analysis techniques were used as the basis of a preventative solution [10]. In this approach, the source code of a web application is inspected to identify the possible SQL queries. The queries are collected prior to the insertion of user input creating a control query. The solution then dynamically monitors for queries being generated at runtime. These queries are then processed by an attribute removal algorithm that removes all data from the query that is contained within quotes as these attributes will have no basis on the syntactic form of the query. This reduced query is then compared using an XOR logic operation to the control query gathered during the static analysis. If this operation returns a result indicating that the two queries are different, the user input must have some form of injection input and it is discarded. This approach is accurate and has very low time complexity as the XOR operation is extremely light on processing. Unfortunately it requires a static analysis which must be accomplished by either the analysis of the web application source code or through the use of a proxy server between the user and the web server.

A framework, using a machine learning approach, implements an Intrusion Detection System that learns the patterns of query strings [11]. It uses a supervised learning training dataset to produce training models. First the strings are parsed into syntactic trees for feature extraction. Feature vectors are used to produce