Context-Sensitive Modeling for Intrusions Detection in Multitier Web Applications

Mounika B, A. Krishna Chaitanya

Abstract- This strategy is mainly focus on to detect intrusion in multitier web applications. Multitier web application include two ends that is front end as well as back end of the applications. Modern multi-tier application systems are generally based on high performance database systems in order to process and store business information. Containing valuable business information, these systems are highly interesting to attackers and special care needs to be taken to prevent any malicious access to this database layer. In this work we propose a novel approach for modeling SQL statements to apply machine learning techniques, such as clustering or outlier detection, in order to detect malicious behavior at the database transaction level. The approach incorporates the parse tree structure of SQL queries as characteristic e.g. for correlating SQL queries with applications and distinguishing benign and malicious queries. We demonstrate the usefulness of our approach on real-world data.

Index Terms: Context-sensitive methods, Intrusion detection system, Machine learning techniques, Multi-tier web architecture, SQL Injection attacks, SQL queries, Pattern mapping techniques

1. INTRODUCTION

Web Services are very much useful nowadays in many domains like banking, travel, social networking etc. These web services operate on the basis of web or internet. These web services are implemented by using front end web server(e.g. http server) and back end server(e.g. database server or file server).Because of popularity of these web services for personal or corporate work, these are always targeted by attackers to do misbehaving activities or attacks[1]. The majority of today's web-based applications does rely on high performance data storage for business processing. A lot of attacks on web-applications are aimed at injecting commands into database systems or try to otherwise trigger transactions to gain unprivileged access to records stored in these systems. See [1] for a list of popular attacks on web applications. Traditional network-based firewall systems offer no protection against these attacks, as the malicious (fractions of) SQL or tampered requests are located at the application layer and thus are not visible to most of these systems. The usual way of protecting modern application systems is by introducing detection models on the network layer or by the use of web application firewall systems. These systems often employ a misuse detection approach and try to detect attacks by matching network traffic or HTTP request against a list of known attack patterns. A very popular system based on pattern matching is for instance the Snort IDS [2].

Another project aiming at the detection of tampered HTTP requests is the ModSecurity module, which provides a rule-engine for employing pattern based rules within a Web-Server [3]. Instead of using pattern based approaches, there exists a variety of papers on employing anomaly-based methods for detecting web-based intrusions [4; 5; 6]. These either try to analyze log-files or protocol-level information to detect anomalies based on heuristics or data-mining techniques. We earlier proposed a rule based learning approach using the ModSecurity module in [7]. These approaches are rooted at the network or application protocol layer. In this work we focus on the detection at the database layer, i.e. the detection of anomalous SQL statements, that are either malicious in the sense that they include parts of injected code or differ from the set of queries usually issued within an application. The main contribution of our work is the use of a grammar based analysis, namely tree-kernel based learning, which became popular within the field of natural language processing (NLP). Our approach incorporates the parse tree structure of SQL queries as characteristic e.g. for correlating SQL queries with applications and distinguishing benign and malicious queries. By determining a context sensitive similarity measure we can locate the nearest legal query for an malicious statements which helps in root cause analysis.

Fig 1 depicts an Intrusion detection system

- *Mounika B is currently pursuing masters degree in Computer Science Engineering in Vardhaman College of Engineering, Shamshabad, Hyderabad, India. PH- 919848789909. E-mail: mouni2089@gmail.com*
- *A Krishna Chaitanya is currently associate professor in Vardhaman College of Engineering, Hyderabad, India. PH-919949122135, E-mail: chaituit2004@gmail.com*

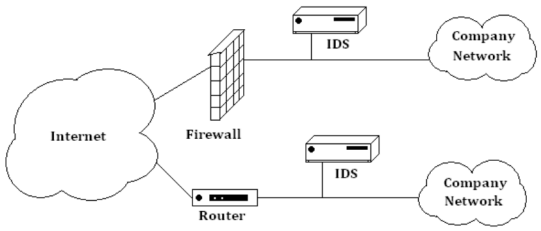

Fig1: Intrusion Detection System

There are following three measures to evaluate efficiency of Intrusion Detection System:

1. Accuracy – Inaccuracy occurs when an IDS signals that an abnormal action is taken in the given environment.
2. Performance – The performance of the system describes the quality of that system. If the performance of IDS is poor then real time detection is not possible.
3. Completeness – When IDS fails to detect an attack then incompleteness occurs. This is very difficult to evaluate because it is impossible to have a global knowledge about all the attacks[3].

2. Related Work

A fundamental rule in three tier architecture is the client tier never communicates directly with the data tier; in a three-tier model all communication must pass through the middle tier called Web tier. Conceptually the three-tier architecture is linear. However, the MVC architecture is triangular: the view sends updates to the controller, the controller updates the model, and the view gets updated directly from the model. Executing malicious statements on a database may result in severe problems, which can range from exposure of sensitive information to losing records or broken integrity. Once an attacker manages to inject code into a database this will likely not only affect specific records, but may lead to a compromise of the complete application environment. This in turn can cause severe outages with respect to data records and a company's public reputation. Although the risk may seem low on a first glance, given the database layer is separated from the public interface (web/presentation layer) and not directly accessible from the outside, anomalous queries caused by e.g. SQL injection attacks are a widespread problem. The Web Hacking Incident Database provides a listing of recent web hacks, a lot of them relying on SQL injections [8]. There have been approaches to apply data-mining and machine learning methods to detect intrusions in databases. Lee et al [9] suggest learning fingerprints of access patterns of genuine database transactions (e.g. read/write sequences) and using them to identify potential intrusions. Typically there are many possible SQL queries, but most of them only differ in constants that represent the user's input. SQL queries are summarized in fingerprints (regular expressions) by replacing the constants with variables or wild-cards. Such fingerprints capture some structure of the SQL queries. Following the approach of [9], queries not matching any of the existing fingerprints are reported as malicious. A drawback of this approach is its inability to correlate and identify fingerprints with applications. In [10] the authors also try to detect SQL injections by a kind of fingerprints. They use parse trees of queries as fingerprints for the queries structure. The main idea here is to compare the parse tree of an SQL statement before and after user-variables have been inserted. Injected SQL fragments will typically significantly change the trees structure. An example of such structural changes in the parse tree of a query is shown in figure 1. In this figure, the rounded nodes of the tree indicate the additional parts that have been added due to the injection SQL fragment ' OR 1 > 0 --. As this work only uses a one-to-one comparison on parse-trees it is missing any generalization capabilities and thus not applicable for machine learning methods, such as clustering and outlier detection. A similar grammar-based approach has been used in [11], which studied the use of syntax-aware analysis of the FTP protocol using tree-kernel methods on protocol parse-trees. A slightly different approach was taken in [12] where the parse tokens are used along with their values to detect anomalies in HTTP-traffic. The latter approach does not use the full parse tree but its leaves. Our work is similar to [11; 12] in the sense that it employs machine learning methods on syntax trees derived from a protocol parser. Also approaches on investigating data dependencies have been proposed in [13] and [14]. Data dependencies refer to access correlations among sensitive data items. Data dependencies are generated in form of classification rules like before an update of item1 a read of item2 is likely. Transactions not compliant to these rules are flagged as malicious. Srivastava et al [14] further distinguish different levels of sensitivity of data items which need to be specified
by hand. Both approaches ignore the structure of SQL queries and are unable to correlate SQL queries with applications. A more recent work has been presented in [15], focusing on the sequential nature of SQL queries. These studies also make use of a smart modeling technique to easily apply data mining methods on their SQL representations.

2.1 SQL Parsing

The basic idea of [10] is to detect SQL injection attacks by means of changes in a queries syntax tree. In order to obtain such a parse tree, a parser for the SQL dialect is required. Usually complex parsers are automatically generated based on a given grammar description using tools such as yacc, antlr or javacc. Unfortunately, the availability of proper grammar descriptions for SQL is pretty sparse and most existing parser implementations are tightly wired into the corresponding DBMS, making it laborious to extract a standalone parser. We therefore decided to modify an existing open-source DBMS, in our case the Apache Derby database, which provides a standalone deployment. The Derby parser is itself generated off a grammar file using javacc, but does not explicitly output a syntax tree suitable for our decomposition. Using the tree-interface of the parser, we derived a tree inspection tool which traverses the tree object of a query and writes out the corresponding node information. SQL Query Victimization To incorporate more syntax, we determine the parse tree of a query. As we are interested in the detection of abnormal queries within our database application, we are looking for a similarity measure for the space of structured objects, i.e. the space of valid SQL parse trees. Thus, we are faced with the problem of having to create a distance function for matching trees. Let q be an SQL query and _q the parse tree of q, identifying with _q the root node of the tree. Each node n within that tree is labeled with an identifier type(n), reflecting the node type. For a node n within we denote by succ(n) the ordered set of successors of n and by succi(n) the ith child of n.

This definition is basically just a formalization of a query's syntax tree. It allows us to enlist the production or grammar rules, which generate a given SQL query q. This list of production rules will be defined as follows: Definition: For a node n within the parse tree _q of a query q, the list of production rules P(n) is given by

$$P(n) = \biguplus_{c \in succ(n)} \{type(n) \rightarrow type(c)\} \uplus \biguplus_{c \in succ(n)} P(c)$$

Given P(n), denote by |P(n)|r the number of times the rule r occurs in P(n). Please note that we use the] notation here for list concatenation, thus, the resulting list may contain the same rule more than once. Now, denoting with Q the set of all valid trees for a given SQL dialect, these simple definitions allow us to define a mapping ' : Q ! Rn, by following the bag of words approach known from text classification tasks like spam detection as proposed in [16].

3. Performance Analysis

The evaluation of the different modeling approaches we collected data of the popular Typo3 content management system. This application heavily depends on the use of SQL for various tasks beyond page content storage, such as session-persistence, user-management and even page-caching.

Table1: capabilities of the different models in Cross Validations

Model	Ratio 200:15			Ratio 1000:15		
	TPR	FPR	time	TPR	FPR	time
3-gram	0.667	0.000	71s	0.667	0.002	643s
4-gram	0.333	0.000	149s	0.733	0.002	1055s
Term vectors	0.667	0.005	2s	0.733	0.002	283s
SQL vectors	0.867	0.000	16s	0.867	0.001	67s

We created a set of distinct queries and added synthetic attacks, which closely reflect modifications that would follow from SQL injections, by inserting typical injection vectors such as OR 'a' = 'a' or the like into legal statements. The intention was to observe whether, using different models, the SVM is to distinguish between legal and malicious statements even though the latter were only marginally different.

Importance of Context- Sensitive

A central question in our work is the importance of contextual information when analyzing SQL queries. We therefore analyzed approaches such as n-grams, term-vector and the SQL victimization described.. In this experiment we did not mean to train a detector, but wanted to explore the expressiveness of the different models and determined the detection rate (TPR) and the false-positive rate (FPR) of the different modeling approaches. As learning algorithm we used an SVM approach within a 10-fold cross-validation. As you can see from table 1, the use of context information results in performance gains especially with respect to the detection rate (TPR) and the fraction of false positives (FPR). This supports our thesis on the importance of the context when analyzing SQL queries. It is worth noting, that the variance in TPR/FPR within the 10-fold cross validation proved to be much smaller for the context-sensitive methods. Additionally, the training time using term- or sql-vectorization decreased due to the smaller number of (irrelevant) attributes. The times in table 1 refer to the

complete parameter-optimization and 10-fold cross-validation process.

4. Conclusion

In this paper we presented two approaches for a context sensitive modeling/ fingerprinting of SQL queries by use of generic models. Using tree-kernels for analyzing SQL statements brings together the results of natural language processing with a highly structured query language. The results confirm the benefit of incorporation of syntax information of previous works [11; 12] in the domain of SQL queries. The consideration of the SQL structures shows performance gains in both performance and speed, the later due to the fewer but far more meaningful features. Compared to previous approaches the tree-kernels allow for a similarity measure on SQL statements providing flexible generalization capabilities. However, a drawback in the use of tree-kernels is their computational overhead. Given a set of 1015 queries, the computation of the kernel matrix took about 210 seconds. Use of hierarchical models, such as hierarchical clustering, may lower the impact of this performance decrease for future detection models.

References

[1] Meixing Le, AngelosStavrou, Brent ByungHoon Kang," Double Guard: Detecting Intrusions in Multitier Web Applications", IEEE Transactions on dependable and secure computing, vol. 9, no. 4, July/august 2012.

[2] F. Valeur, G. Vigna, C. Kru¨ gel, and R.A. Kemmerer, "A Comprehensive Approach to Intrusion Detection AlertCorrelation," IEEE Trans. Dependable and Secure Computing, vol. 1, no. 3, pp. 146-169, July-Sept. 2004.

[3] Openvz, http://wiki.openvz.org, 2011.

[4] Joomlacms, http://www.joomla.org/, 2011.

[5] http://www.dummies.com/how-to/ content/ examining-differenttypes- of-intrusion-detection s.html

[6] http://advanced-network-security .blogspot .in/2008/04/threemajor- types-of-ids.html

[7] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications," Proc. Int'l Symp. Recent Advances in Intrusion Detection (RAID '07), 2007.

[8] Karen scarfone,Petermell,"Guide to Intrusion Detection and Prevention Systems (IDPS)" , NIST National institute of standards & Technology (Technology Administration U.S. Department of commerce , Special Publication 800-94

[9] http://www.omnisecu.com/ security/ infrastructure- and-emailsecurity/ types- of-intrusion-detection-systems.htm.

[10] http://en.wikipedia.org/ wiki/ Multitier _architecture#Comparison_with_MVC _architecture.

[11] R. Gerstenberger. Anomaliebasierte Angriffserkennung im FTP-Protokoll. Master's thesis, University of Potsdam, Germany, 2008.

[12] P. D¨ussel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In Proc. of Int. Conf. on Information Systems Security (ICISS), pages 188–202, 2008.

[13] Y. Hu and B. Panda. A data mining approach for database intrusion detection. In Proc. of ACM SAC, pages 711–716. ACM, 2004.

[14] A. Srivastava, S. Sural, and A. K. Majumdar. Database intrusion detection using weighted sequence mining. JCP, 1(4):8–17, 2006.

[15] A. Roichman and E. Gudes. DIWeDa - detecting intrusions in web databases. In Proc. of IFIP Conf. on Data and Appl. Security, pages 313–329. Springer, 2008.

[16] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In Machine Learning: ECML-98, pages 4–15. Springer, 1998.

[17] K. Rieck, T. Holz, C. Willems, P. D¨ussel, and P. Laskov. Learning and classification of malware behaviour. In Proc. of DIMVA. Springer, 2008.

[18] D. Haussler. Convolution kernels on discrete structures. Technical report, Dept. of Computer Science, UC Santa Cruz, 1999.

[19] M. Collins and N. Duffy. Convolution kernels for natural language. In Advances in Neural Information Processing Systems 14, pages 625–632. MIT Press, 2001.

[20] G. D. Zhou, M. Zhang, D. H. Ji, and Q. M. Zhu. Tree kernelbased relation extraction with context-sensitive structured parse tree information. In Proc. of Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 728–736. Assoc. for Computer Linguistics, 2007

[21] Open Web Application Security Project. The Top list of most severe web application vulnerabilities, 2004.

[22] M. Roesch. Snort: Lightweight intrusion detection for networks. In Proc. of LISA, pages 229–238. USENIX, 1999.