Volume 3, Issue 12, December 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Hybrid Approach for handling SQLI Vulnerabilities in Web Applications

M. Nirupama Bhat*, N. Veeranjaneyulu, A. Raghunath

School of Computing
Vignan's University
Guntur, India

Abstract— Huge volumes of information storage and transfer is conveniently performed through internet using web applications. The web applications are highly vulnerable to injection flaws through which the malicious user/attacker can attack a system. SQLIA is a wide spread injection attack where the malicious user finds the parameter through which he enters the database and corrupt or destroy its contents. This paper discusses about the web application architecture, probable injection vulnerabilities likely to occur in various databases. A hybrid approach to detect and prevent SQLIA is proposed. It also compares the performance with respect to response time and detection trend.

Keywords— Injection Attacks, Web applications, vulnerabilities, prevention and detection and Databases.

I. INTRODUCTION

Web applications are applications that can be accessed over the internet by using any web browser that runs on any operating system and architecture. They have become ubiquitous due to the convenience, flexibility, availability and interoperability that they provide. Unfortunately, Web applications are also vulnerable to a variety of threats. Injection flaws which are easy to detect and exploit, allow attackers to relay malicious code through a web application to another system. SQL injection is one such attack, frequently employed by malicious users for different reasons, e.g. financial fraud, theft confidential data, or simply for fun. These attacks include calls to the operating system via system calls, external programs via shell commands, as well as calls to backend databases via SQL. SQL Injection Attack (SQLIA) is a particularly widespread and dangerous form of injection. To exploit a SQL injection flaw, the attacker finds a parameter that the web application passes through to a database. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

SQL Injection Attacks are one of the most significant of such threats. Web applications are at highest risk to attack since often an attacker can exploit SQL injection vulnerabilities remotely without any database or application authentication. Injection attacks can be very easy to discover and exploit, but they can also be extremely obscure. Most application developers underestimate the risk of SQL injections attacks against applications that use back-end database. SQL Injection Attacks can also escape traditional tools such as firewalls and Intrusion Detection Systems (IDSs) [4] because they are performed through ports used for regular web traffic which usually are open in firewalls. On the other hand, most IDSs focus on the network and IP layers whereas SQLIAs work at application layer. Researchers have proposed a range of techniques and tools to help developers to compensate the shortcoming of the defensive coding.

II. WEB ARCHITECTURE

Web application, or webapp, is the general term that is normally used to refer to all distributed web-based applications [9]. Web Applications are similar to computer-based programs but differ only in that they are accessible through the web, allowing the creation of dynamic websites and providing complete interaction with the end-user. Web Applications are placed on the Internet and all processing is done on the server, the computer which hosts the application. These applications are sets of web pages, files and programs that reside on a company's web server, which any authorized user can access over a network such as the World Wide Web or a local intranet.

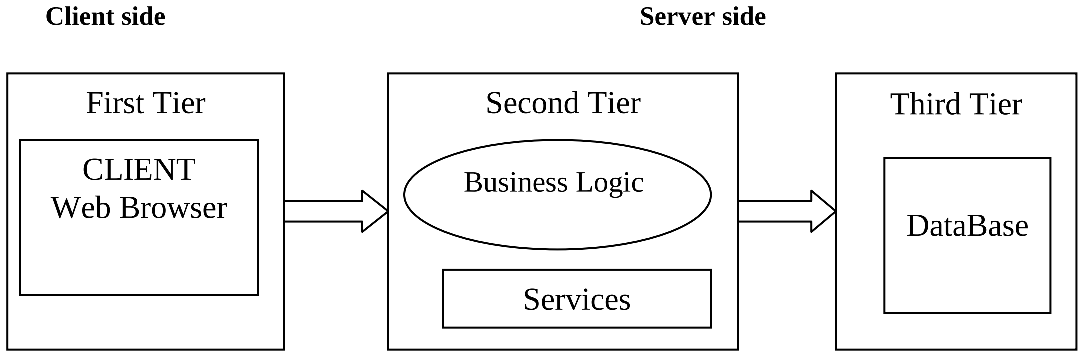


Fig 1: Three Tier J2EE Web Applications Model

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. The most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, JSP/Java, PHP, Perl) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

Normally, the first tier is a Web browser on the client side, the second is the real engine on the server-side where the applications core runs, and the third layer is a database as shown in Figure 1.1. A server processes all user transactions and usually the end-user simply accesses the web application by a Web browser, interacting with it. Since web applications reside on a server, they are easy to manage.

Need to Secure Web Applications: Security breaches through **w**eb application must be prevented as web application vulnerabilities are the main causes of any kind of attack. Web applications (shopping carts, forms, login pages, dynamic content, and other bespoke applications) are designed to allow the website visitors to retrieve and submit dynamic content including varying levels of personal and sensitive data. If these web applications are not secure, then the entire database of sensitive information is at serious risk. A Gartner Group study reveals that 75% [10] of cyber attacks are done at the web application level. This happens because

- Websites and web applications are easily available via the internet 24 hours a day, 7 days a week to customers, employees, suppliers and therefore also hackers.
- Firewalls and SSL provide no protection against web application hacking, simply because access to the website has to be made public.
- Web applications often have direct access to backend data such as customer databases.
- Most web applications are custom-made and, therefore, involve a lesser degree of testing than off-the-shelf software. Consequently, custom applications are more susceptible to attack.

SQL Injection vulnerability is a common weakness of database-driven web sites. This flaw is easily detected and easily exploited, and as such, any site or application with even a minimal user base is highly likely to be subject to an attempted attack of this kind. Moreover, SQL Injection functions independent of program language, platform, architecture, operating system, database, third-party applications and network/computer configurations. This means that almost all SQL databases and programming languages are potentially vulnerable.

SQL Injection is an input validation problem[5] that has to be considered and programmed by the web application developer. Every web application based on a database, if not perfectly developed, could be exploited by a SQLIA. The technologies vulnerable to this attack are dynamic script languages including ASP, ASP.NET, PHP, JSP, and CGI. Examples of relational databases include Oracle, Microsoft Access, MS SQL Server, MySQL, DB2, all of which use SQL as their basic building blocks. In short, SQL Injection is a dangerous vulnerability and all programming languages and all SQL databases are potentially vulnerable. The characteristics of SQLIA is given in table 1.

Table 1: Characteristics of SQL Injection Attack

SQL Injection Attack			
Weakness Prevalence	High	Consequences	Data Loss, Security bypass
Remediation cost	Low	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

III. DATABASE FOOT PRINTING

Foot printing reveals the vulnerabilities and improves the ease with which they can be exploited. It is the process or accumulating data regarding a specific network environment usually for the purpose of finding ways to improve in the environment. There are two kinds of databases: open source and commercial. All of them allow and require different SQL syntax. Determining the database engine type is fundamental to continue with the injection attack. The easiest way to get this information is by error messages (ODBC will normally display the database type as part of the driver information when reporting an error). The other way to obtain useful information is by using specific characters, commands, stored procedures and syntax[8]. An educated guess based on the operating System and Web Server gives more certainty about the SQL databases that are being used in the applications. Some useful commands that can be used to determine what databases [5] are being used are given in the table 2&3.

Table 2: Database Foot Printing : differences among various databases.

	MS SQL	MySQL	ACCESS	ORACLE	DB2
UNION	✓	✓	✓	✓	✓
SubSelects	✓	✓	X	✓	✓
Batch Queries	✓	X	X	X	X
Stored Procedures	✓	X	X	✓	X
Linked DBs	✓	✓	X	✓	✓

Table 3 : Database Foot Printing: useful commands to determinate the database

	MSSQL	MySQL	Access	Oracle	DB2
Concat	'+'	concat (" , ")	"&"	'‖'	"+"
Null replace	Isnull()	Ifnull()	Iff(Isnull())	Ifnull()	Ifnull()
Position	CHARINDEX	LOCATE( )	InStr( )	InStr( )	InStr( )
OS Interact-ion	xp_cmdshell	select into outfile/dumpfile	#date#	utf_file	import from export to
Cast	Yes	No	No	No	Yes

IV. HYBRID APPROACH

A Combination of approaches are used for protecting Web applications against SQL injection, incorporating the uniqueness of Signature based method and auditing method. From signature based method standpoint of view, we presents a detection mode for SQL injection using token wise sequence alignment of the query. On the other hand from the Auditing based method standpoint of view, it analyzes the transaction to find out the malicious access. In the Signature based method we uses an approach called Hirschberg algorithm[1], it is a divide and conquer approach to reduce the time and space complexity. This system was able to stop all of the successful attacks and did not generate any false positives.

Signature Based Approach:

Our approach against SQLIAs is based on Signature based approach[1], which has been used to address security problems related to input validation. This approach has three modules which are used to detect the security issues. They are 1. Monitoring module 2. Specifications module 3. Analysis module. Monitoring module has got the statement from the web application which can decide whether it can send the statement to database for execution. Analysis module uses Hirschberg algorithm to compare the statement from the specifications. Specifications comprise the predefined keywords and send it to analyze module for comparisons. It analyzes the comparisons as well as database transaction. If it finds any suspicious activity, it acts as an active agent to stop the transaction and audit the attacks. If both analysis module and auditing module has satisfied, it provides the complete transaction. The following figure clearly depicts the architecture of the system to prevent the SQL Injection attacks using new combined approach.

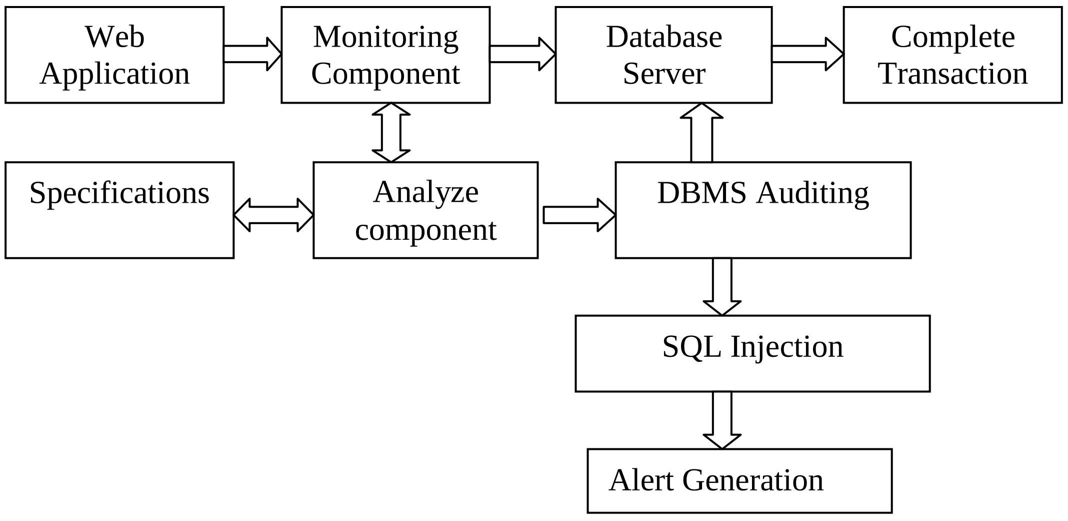


Fig 2: Hybrid Approach for Preventing SQL Injection

Algorithm

We are using Hirschberg's algorithm, named after its inventor, Dan Hirschberg [6]. It is a dynamic programming algorithm[14] that finds the least cost sequence alignment between two strings, defined to be the sum of the costs of insertions, replacements, deletions, and null actions needed to change one string to the other. Hirschberg's algorithm is simply described as a divide and conquer version of the Needleman-Wunsch algorithm[6]. Hirschberg's algorithm is commonly used in computational biology to find maximal global alignments of DNA and protein sequences and this proposal incorporated this algorithm here to find out the similarities. Hirschberg's algorithm is a generally applicable algorithm for finding an optimal sequence alignment. The formula to plot the values is shown in the following table.

Table 4 : Hirschberg Algorithm

F(i, j)	A	C	T	G
A	F(i-1, j-1)	F(i,j-1)+p p-gap penalty		
C	F(i-1,j)+p p-gap penalty			
T				

F(i,j)= Max{F(i-1,j-1)+s($X_{i,}$ Y_i), F(i,j-1)+p, F(i-1,j)+p)}
s($X_{i,}$ Y_i)- score for aligning the characters at positions i and j, p is the penalty for a gap
F(i,j) is a type of running best score as the algorithm moves through every position in the matrix.
But in our approch, gap penalty is ignored. If Xi=Yi, then plot 1 else 0 till all the character has been visited.The algorithm directly match between the two sequences. For two sequences a= a1 a2….. an and b= b1 b2….bn, where sij= s(a1 a2……ai, b1 b2 ……bj) where sij is the score at position i in sequence a and position j in sequence b, s(ai bj) is the score for aligning the characters at positions i and j, p is the penalty for a gap of length x in sequence a, and p is the penalty for a gap of length y in sequence b.
In our proposed system, it maintains the table as it is like gene comparison that it contains keyword which are present in horizontal or vertical line and it will compare the incoming tokens with this predefined values using this algorithm for identity. As per the Hirschberg algorithm, it divides the token and it checks the each token with the predefined tokens using divide and conquer methodology.
Example, take a sample URL parameter attacks,
http://www.estore.com/bookdetail.jsp?tr=100&status=read
This has been modified in the following way, where the status variable is changed in such a fashion that it deletes the page instead of reading.
This has been shown in the following URL address as "status=del".
http://www.estore.com/bookdetail.jsp?tr=147&status=del

The following table clearly explains the approach system to prevent such attacks. Hirschberg algorithm uses divide and conquer approach in order to reduce time and space complexity. X is the predefined statement which has been stored in specification module and Y is the statement which has been get it from monitoring module and it compares using this algorithm. It divides the problem into two sub problems, from tr to status and estore to bookdetail.jsp. First it compares the first sub problems and it find out the attacks and it won't go for other sub problems and it sends the notification to monitoring module to stop the transaction and audit the attacks.

Table 5: Hirschberg algorithm to find out attacks

X Y	estore	bookdetail.jsp	tr	&	Status=read
estore					
bookdetail.jsp					
		Divide	And	Conquer	
Tr					
&					
Status=del					

Procedure using Tool

Achieve a complete and common methodology is not a trivial task. It is a long and complex process that considers different aspects and features of the evaluated tool. A first reason is that each tool is different and independent to the others, so it requires specific configurations and tests. Moreover there are several parameters to analyzes systematically and each one with different approaches, network configurations and objectives. We provide a general methodology adaptable to any types of security tools against SQLIAs. An abstract overview of our approach, focusing on the main phases of our methodology and afterwards we describe in details each step of the proposal procedure. Finally we will discuss our proposal idea.

Abstract Methodology Diagram

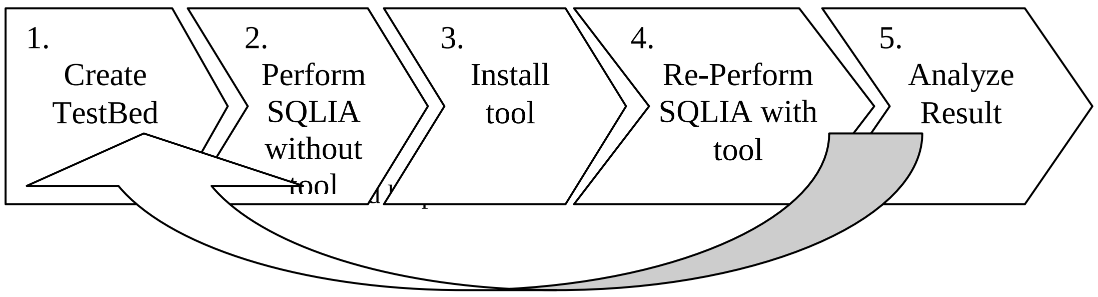


Fig 3: Proposal Evaluation Methodology: General Model

V. DISCUSSION AND RESULTS

Analysis of the Implementation

Here, we will first detect the SQL injection attack using tool and also using Hirschberg algorithm. To detect all types of attacks we have to implement both approaches. So many tools are available to find out the web application attacks by scanning the entire application. From that we choose the best tool called Acunetix web vulnerability scanner[8] which itself acts as a tool and scanner. We will follow the above mentioned procedure to find out the attacks and compare the results with manual approach. In this paper, we proposed a combination of approaches to prevent the SQL Injection attack. The steps include 1) identifying hot spot from the application, 2) Hirschberg algorithm to find out the SQL Injection attacks. 3) Using DBMS auditing methods to find out the transactions. Hirschberg algorithm uses divide and conquer approach to detect the SQL Injection attacks in order to reduce the time and space complexity and it provides the complete execution after analyzing the DBMS auditing.

Detection of SQL injection types is done using character level checking and syntax level checking [3] and display the results when such attacks found. For Prevention we used Signature based method, Auditing method and Code review method. Signature method is based on Hirschberg algorithm, which follows divide and conquer approach. We will locate the hotspots, and then divide the query into tokens. Compare the tokens of predefined query and the dynamically generated query using divide and conquer approach. In our proposed system, it maintains the table as it is like gene comparison that it contains keywords which are present in horizontal or vertical line and it will compare the incoming tokens with this predefined values using this algorithm for identity.

This algorithm mainly used for reducing the time and space complexity of the system. For example, x and y are strings to be compared, where $|x| = n$ and $|y| = m$. The Needleman-Wunsch algorithm[14] finds an optimal alignment in O(nm) time, using O(nm) space. Hirschberg's algorithm is a clever modification of the Needleman-Wunsch Algorithm which still takes O(nm) time, but needs only O(n+m) space for comparing these two x and y strings.

Table 6: using Hirschberg Algorithm to find the similarities

(i,j)	Select	*	From	Prod	Where	User	=	“		“	And	pass	=	“		“	;
Select	1																
*		1															
From			1														
Prod				1													
Where					1												
User						1											
=							1										
“								1									
					Divide	And	Conquer										
“										1							
And											1						
Pass												1					
=													1				
“														1			
“																1	
;																	1

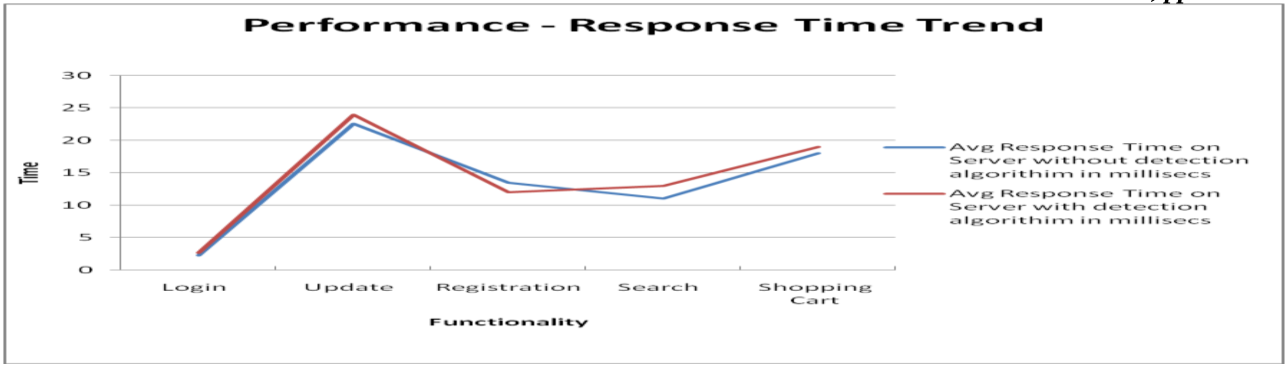
In this SQL statement, this is an hot spot which has been identified by the analyze module and it send to this table for detect and prevent SQL injection attacks.

SQL= ”select * from prod where user= ‘“&tusername&”’ and pass =   ’”&tpassword&”’”;

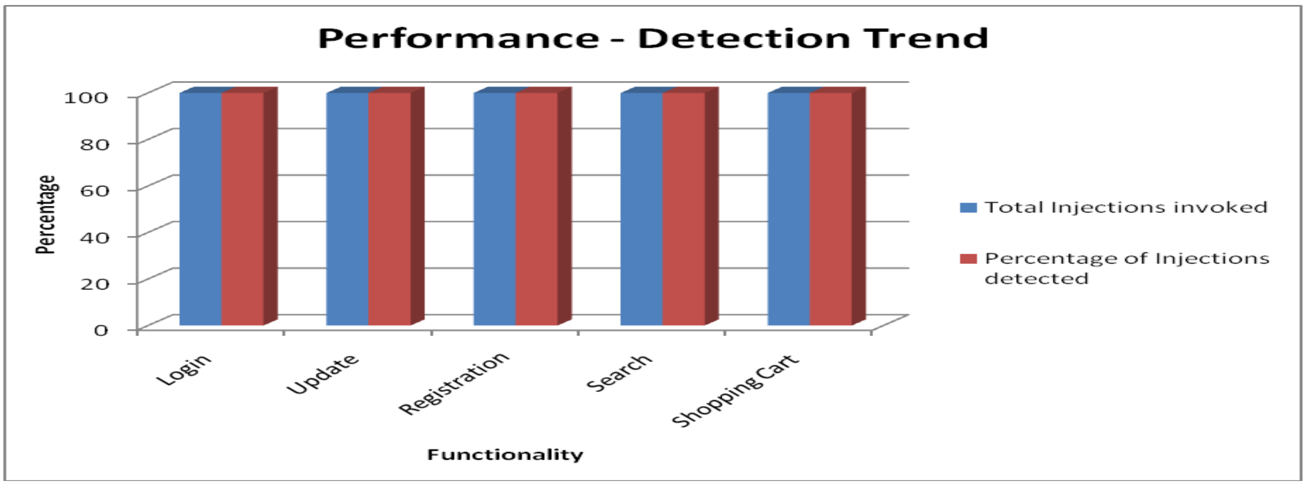
As per the Hirschberg algorithm, it divides the token and it checks the each token with the predefined tokens using divide and conquer methodology.

We took four web applications that are known to be vulnerable. Three of them are commercial applications. They are Employee Directory, Bookstore and Classifieds. One application is developed by us to show the type of attacks and type of approach used. And the other is open source applications purposely insecure Web application that was developed by the Open Web Application Security Project and asked us to find out vulnerabilities. We tried to analyze the performance of the change in terms of response times and detection capability. We identified key functional areas in the applications and subjected those to the SQL injections. We noted the time taken for SQL execution on server logs with and without the detection algorithm.

The data results are captured in the below diagrams and then show a good trend in the response time and detection capability without adding any undue load to the application and detecting and preventing all the vulnerabilities.



Graph 1: Performance – Response Time Trend



Graph 2: Performance – Detection Trend

VI. CONCLUSION AND FUTURE WORK

In this paper, we first identified the various types of SQLIAs. Then we investigated on SQL injection detection and prevention techniques. After that we compared these techniques in terms of their ability to stop SQLIA. Our approach also provides advantages over the many existing techniques whose application requires customized and complex runtime environment. Since it defined at the application level, requires no modification of the runtime system, and imposes a low execution overhead. In future work, we will find out the techniques to find out SQL injection vulnerabilities in stored procedures. And also similar technique can be developed to address different kinds vulnerabilities like cross scripting.

REFERENCES

[1] R. Ezumalai, G. Aghila. “Combinatorial Approach for Preventing SQL Injection Attacks”. IEEE International Advance Computing Conference. Pages 1212 – 1217, March 2009.
[2] William G.J. Halfond, Alessandro Orso,Panagiotis Manolios, “A Classification of SQL Injection Attacks and Countermeasures”, College of Computing Georgia Institute of Technology IEEE, 2006.
[3] Konstantinos Kemalis and Theodoros Tzouramanis. SQL-IDS: A Specification-based Approach for SQL Injection Detection Symposium on Applied Computing. 2008, Pages: 2153-2158 , Fortaleza, Ceara, Brazil. New York, NY, USA: ACM.
[4] Ashish kamra, Elisa Bertino, Guy Lebanon, "Mechanisms for database intrusion detection and response", Data security & privacy, Pages 31-36, ACM, 2008.
[5] "OWASP – Open Web Application Security Project. Top ten most critical web application vulnerabilities," 2010. http://www.owasp.org/
[6] Wikipedia, Needleman Wunsch. http://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm
[7] A. Dasgupta, V. Narasayya,M. Syamala. A Static Analysis Framework for Database Applications. IEEE 25th International Conference on Data Engineering. Pages 1403 – 1414, March 2009.
[8] Acunetix Web Vulnerability Scanner,2008, http://www.acunetix.com/vunerability-scanner
[9] Wikipedia, "web application", http://en.wikipedia.org/wiki/Web-Application
[10] /Vulnerability Type Distributions in CVE. cwe.mitre.org, Oct 2006.
SANS Institute, http://www.sans.org/whatworks http://cwe.mitre.org/documents/vuln-trends.html