CHAPTER 2
CHAPTER 2
LITERATURE SURVEY

The SQL Injection Attack detection and prevention techniques can be classified into the following levels, Defensive Coding Practices, Detection and Prevention Techniques, General Techniques against SQLIA’s, Automatic Approach, Combination of Static and Runtime Analysis, Cryptographic Approaches and Intrusion Detection Techniques.

2.1 DEFENSIVE CODING PRACTICES

Huang et al [30] proposed a tool named WEBSSARI which describes an approach to ensure Web application security. Although defensive coding practices remain the best way to prevent SQL Injection Vulnerabilities, their application is problematic in practice. It does not provide an automated mechanism for detection and prevention of SQL Injection Attack. Defensive Coding is prone to human error and it is not rigorously applied as an automated techniques. Many developers make an effort to code safely. It is extremely difficult to apply defensive coding practices rigorously and correctly to identify all sources of input. In fact, many of the SQL Injection Vulnerabilities discovered in real applications are due to human errors. If developers forget to perform adequate input validation for all the approaches of this subsection generate a significant number of false negatives.

Scott et al [63] used a proxy to filter input and output data streams for a web application even though this technique can be effective against SQLIA. It requires developers to correctly specify filtering rules for each application input. This step of the process is prone to human error and leaves the application vulnerable if the developer has not adequately identified all injection points and correctly expressed the filtering rules. Like defensive coding practices this techniques cannot provide guarantees of completeness and accuracy.
Anley [5] and Howard et al [28] described that the developers have employed a range of code-based solutions to counter SQLIA’s. Input validation based techniques include checking user input for keywords, identifying known malicious patterns and escaping potentially troublesome characters. While these techniques can stop straightforward and unsophisticated attacks, attackers have learned to use alternate encoding schemes such as hexadecimal, ASCII and Unicode to obfuscate their attacks. In addition, simply checking user input for malicious keywords would clearly result in a high rate of false positives, since an input field could legally contain words that match SQL keywords (i.e. “FROM”, “OR”, “AND”). Another widely proposed coding solution is to use stored procedures for database access. The ability of stored procedures to prevent SQLIA’s is dependent on its implementation.

McDonald [50] [51] discussed that the defensive coding has not been successful in completely preventing SQLIA. Attackers keep finding new attack strings or subtle variations on old attacks that can avoid the checks programmers put in place. This improved coding practice can help to mitigate the problem. This approach is being fully automated, can provide stronger guarantees about the completeness and accuracy of the protections put in place. Intrusion Detection System (IDS) or Intrusion Prevention System (IPS) should never be used alone to protect applications from SQL Injection vulnerabilities. Rather, they should be implemented as alerting mechanisms.

2.2 STATIC ANALYSIS TECHNIQUES

Christensen et al [14] proposed a string analysis technique using Java. This technique performs a conservative string analysis of an application and creates automata that express all the possible values of a specific string can have at a given point in the application. This technique constructs a flow graph that abstracts away the control flow of the program and represents string-manipulation operations performed on string variables. This technique analyzes the flow graph and simulates the string manipulation operations that are performed on the string. The result of the analysis is a Non-Deterministic Finite Automaton (N DFA) that
expresses, at the character level, all the possible values the considered string can assume at the hotspot. The string analysis is conservative, if a program error is detected then the author illustrate that the extensive benchmarks demonstrating the analysis is efficient and produces results of useful precision.

Gould [22] [23] presented a sound and static program analysis technique to verify that dynamically generated query strings which do not contain type errors. Gould describe that the analysis technique and provide soundness results for the static analysis algorithm. The prototype tool is based on the algorithm and present several illustrative defects found in senior software-engineering student-team projects. JDBC-Checker is a technique for statically checking the type correctness of dynamically-generated SQL queries. JDBC-Checker is able to detect one of the root causes of SQLIA vulnerabilities in code improper type checking of input. However, the author’s technique would not catch more general forms of SQLIA’s because most of these attacks consist of syntactically and type correct queries.

Livshits [43] proposed a static analysis approach based on a scalable and precise point to analysis. In this system, user-provided specifications of vulnerabilities are automatically translated into the static analyzers. This approach finds all vulnerabilities matching a specification in the statically analyzed code. Context sensitivity, combined with improved object naming, proved instrumental in keeping the number of false positive is low.

Penetration testing and static code analysis are two techniques used for the detection of security vulnerabilities. Nuno Antunes [7] compared the effective of these two techniques on the detection of SQL Injection vulnerabilities in web services code. The strengths and limitations of these techniques shown by, several commercial and open source tools are used to detect vulnerabilities in a set of vulnerable services. Results suggest that, static code analyzers are able to detect more SQL Injection vulnerabilities than penetration testing tools.
Jovanovic et al [33] [34] presented a novel, precise alias analysis targeted at the unique reference semantics commonly found in scripting languages. Moreover, this technique enhances the quality and quantity of the generated vulnerability reports by employing a novel, iterative two-phase algorithm for fast and precise resolution of file inclusions. This work addresses the problem of vulnerable web applications by means of static source code analysis. The drawback of this technique can generate false positives. Moreover, this technique relies on declassification rules to transform untrusted input into safe input and it can also generate false negatives.

Yichen Xie [81] presented a static analysis algorithm for detecting security vulnerabilities in PHP (Hypertext Preprocessor), a popular server-side scripting language for building web applications. This analysis uses novel three-tier architecture to capture information at decreasing levels of granularity at the intrablock, intraprocedural and interprocedural level. This architecture enables us to handle dynamic features of scripting languages that have not been adequately addressed by previous techniques. These approaches scan the application and leverage information flow analysis or heuristics to detect code that could be vulnerable to SQLIA’s. Because of the inherently imprecise nature of the static analysis these techniques can generate false positives. Since they rely on declassification rules to transform untrusted input into safe input, they can also generate false negatives.

Stuttard [68] proposed a static code analyzer to analyze the code without actually executing it. The analysis performed by existing tools varies depending on their sophistication, ranging from tools that consider only individual statements and declarations to others that consider the complete code. Among other usage (e.g., model checking and data flow analysis), these tools provide an automatic way for highlighting possible coding errors.

William G.J. Halfond [80] discussed about the parameter mismatches can cause serious errors in the web application and are difficult to identify using
traditional testing and verification techniques. To address this problem, this work proposes a static-analysis based approach for identifying parameter mismatches. This approach works by (1) Identifying the set of interfaces accepted by each web component. (2) Determining the set of interface invocations performed by each web component. (3) Checking whether each interface invocation matches an accepted interface of the invocation target.

Arjun Dasgupta [8] presented a framework that adapts traditional program analysis by leveraging understanding of data access APIs in order to identify the malicious attacks such as SQL injection detection, workload extraction, identifying performance problems and verifying data integrity constraints in the application. This framework consists of a core set of static analysis tools for database applications can significantly enhance the ability for developers to identify security, correctness and performance problems in the application during the development phase of the application lifecycle.

Martin et al [47] developed both static and dynamic techniques to find solutions to PQL queries. This static analyzer finds all potential matches conservatively using a context-sensitive, flow-insensitive, inclusion-based pointer alias analysis. Static results are also useful in reducing the number of instrumentation points for dynamic analysis. The dynamic analyzer instruments the source program to catch all violations precisely as the program runs and to optionally perform user-specified actions.

Kc et al [36] described a new and general approach for safeguarding systems against any type of code-injection attack. This technique applies Kerckhoff’s principle, by creating process-specific randomized instruction sets like machine instructions of the system executing potentially vulnerable software. An attacker does not know the key to the randomization algorithm will inject code that is invalid for that randomized processor causing a runtime exception. This approach is equally applicable against code-injecting attacks in scripting and interpreted languages. e.g., web-based SQL injection.
2.3 COMBINATION OF STATIC AND RUNTIME ANALYSIS

Muthuprasanna et al [54] proposed a technique, which combines static application code analysis with runtime validation to detect the occurrence of SQL Injection Attack. The deployment of this technique eliminates the need to modify source code of application scripts, additionally allowing seamless integration with currently-deployed systems. This technique does not need code modification and simple web server patch required. Using SQL-graphs optimized runtime analysis can be evaluated in static analysis method and SQL query validation executed in parallel, for faster webpage accesses. SQLIA’s using access control violations in the script and different character encodings are also captured. This paper provides various optimizations improving overall efficiency.

Wei et al [79] proposed a novel technique to defend against SQL Injection Attacks targeted at stored procedures. This technique combines static application code analysis with runtime validation to eliminate the occurrence of such attacks. In the static part, Wei et al design a stored procedure parser and for any SQL statement which depends on user inputs explain how an attacker could exploit this vulnerability. This paper presents various techniques that can be employed to gain illegitimate access to the system. In static level validation SQL graph is explained in dynamic level. Each SQL string is enclosed by the SQLIACHECKER() function which does the runtime validation and legitimacy check SQLI as well as the network resources. This work proceeds by using an explicit instrumentation of source code can be avoided.

2.4 AUTOMATIC APPROACH

Antunes et al [6] proposed a new automatic approach for the detection of SQL Injection and XPath Injection Vulnerabilities, this representative workload is used to exercise the web service and a large set of SQL/XPath Injection Attacks are applied to disclose vulnerabilities. Vulnerabilities are detected by comparing the structure of the SQL/XPath commands issued in the presence of attacks to the ones previously learned when running the workload in the absence of attacks.
Halfond et al [27] proposed a new highly automated approach for dynamic detection and prevention of SQLIA’s. Intuitively, this work is by identifying “trusted” strings in an application and allowing only these trusted strings to be used to create the semantically relevant parts of a SQL query such as keywords or operators. Dynamic tainting is a mechanism used to implement this approach. The advantage of this approach is highly automated and requires minimal intervention does not require additional infrastructure and can be automatically deployed.

Stephen Thomas et al [66] proposed an automated method for removing SQL Injection Vulnerabilities from JAVA code by converting plain text SQL statements into prepared statements. Prepared statements restrict the way that input can affect the execution of the statement. An automated solution allows developers to remove SQL Injection Vulnerabilities by replacing the vulnerable code with generated secure code. The prepared statement separates the values in a query from the structure of SQL. The programmer defines a skeleton of an SQL query and then fills in the holes of the skeleton at runtime. The prepared statement makes it harder to inject SQL queries because the SQL structure cannot be changed. To use the prepared statement, the web application has to modify entirely; all the legacy web application must be re-written to reduce the possibility of SQL injections.

Kosuga [38] proposed an SQLIA vulnerability testing tool, Sania: Syntactic and semantic analysis for automated testing against SQL Injection, identifies injectable parameters by comparing the parse trees and HTTP responses for a benign HTTP request and the corresponding auto-generated attack. Sania, designed to check for SQL injection vulnerabilities in the development and debugging phases is focused only on two parameters: efficiency and false positive rate. The main drawback of this approach is the high rate of false positives (about 30%) and the need for application developers to be involved in the SQLIA vulnerability testing.
MeiJunjin [52] proposed an automated prototype tool SQLInjectionGen for SQL Injection Vulnerability detection. This approach is compared to static analysis for identifying true SQL injection vulnerabilities. SQLInjectionGen had no false positives, but had a small number of false negatives while the static analysis tool had a false positive for vulnerability that was actually protected by a white or black list.

Vieira et al [75] discussed that a penetration testing tools provide an automatic way to search for vulnerabilities. Vieira et al explains an empirical study using four commercial vulnerability scanners to detect security vulnerabilities for publicly available web services. A large number of vulnerabilities were observed, confirming that many services are deployed without proper security testing. A final observation is that SQL Injection Vulnerabilities are prevalent in the web services tested as they represent more than 84% of all vulnerabilities detected.

Kals [35] proposed a tool SecuBat, demonstrates that how the attackers automatically discover and exploit application-level vulnerabilities in a large number of web applications. To prevent the attacks, SecuBat, a generic and modular web vulnerability scanner is developed. It automatically analyzes web sites to find the exploitable SQL Injection and XSS Vulnerabilities. SecuBat is used to find many potentially vulnerable web sites. These tools perform attacks against web applications, usually in a black-box fashion and detect vulnerabilities by observing the application’s response to the attacks. However, without exact knowledge about the internal structure of applications, a black-box approach might not have enough test cases to reveal existing vulnerabilities and also have false positives.

Paros [58] developed a free tool for testing web application vulnerabilities without rewriting any scripts in application. This tool automatically scans for SQL Injection Vulnerabilities with pre-defined attack codes. Paros checks the contents
of HTTP response messages to determine whether an SQL Injection Attack was successful or not.

Wassermann [78] addressed the limitations by proposing a precise, sound and fully automated analysis technique for SQL Injection. This technique avoids the attacks by considering those queries for which user input changes the intended syntactic structure of the generated query. The context free grammar is used to conformance to this policy.

Geer [21] discussed as a bot is a program that operates automatically as an agent for a user or another program. Hacker’s forward bots to victims by a number of means and the software automatically infects vulnerable computers. The bots then wait for commands from a hacker and can manipulate them in the infected systems without the user’s knowledge. There are some major threads in web application security which can break the web application security by using bots.

Fred Dysart [20] presented specifically targets input validation vulnerabilities found in SQL queries that may lead to SQL Injection Attacks (SQLIA’s). Fred Dysart introduces a tool that automatically detects and suggests fixes to SQL queries that are found to contain SQL Injection Vulnerabilities (SQLIV’s).

Anh Nguyen-Tuong [4] presented a fully automated approach to securely hardening web applications. This technique is based on precisely tracking taintedness of data and checking specifically for dangerous content only in parts of commands and output that came from untrustworthy sources. Unlike previous work in which everything that is derived from tainted input is tainted, this approach precisely tracks taintedness within data values.

2.5 DETECTION AND PREVENTION TECHNIQUES

Huang et al [29] discussed the use of a number of software testing techniques such as dynamic analysis, black-box testing, fault injection, behavior
monitoring and suggest the mechanisms for applying these techniques to Web applications. Author named the tool as WAVES (Web Application Vulnerability and Error Scanner), a black-box technique for testing Web applications for SQL Injection Vulnerabilities. The technique uses a Web crawler to identify all hotspots in a Web application that can be used to inject SQLIA’s. It then builds attacks that target such hotspots based on a specified list of patterns and attack techniques. WAVES monitors the application’s response to the attacks and uses machine learning techniques to improve its attack methodology. This technique improves over most penetration-testing techniques by using machine learning approaches to guide its testing. However, like all black-box and penetration testing techniques, it cannot provide guarantees of completeness.

Wassermann [77] presented the design of a static analysis framework to address these input related problems in the context of web applications. In this framework, the abstract model is used for a source program that takes user inputs and dynamically constructs SQL queries. Framework applies some novel checking algorithms and it automatically indicates or verifies the absence of security violations in the original application program. This approach uses static analysis combined with automated reasoning to verify that the SQL queries generated in the application layer cannot contain a tautology. The primary drawback of this technique is that its scope is limited to detecting and preventing tautologies and cannot detect other types of attacks.

Halfond et al [26] proposed a novel technique to counter SQL Injection. The technique combines conservative static analysis and runtime monitoring to detect and stop illegal queries before they are executed on the database. This technique builds a conservative model of the legitimate queries in static part. The dynamically generated queries are inspected by using statically-built in model dynamic part. The primary limitation of this technique is that its success is dependent on the accuracy of its static analysis for building query models. Certain types of code obfuscation or query development techniques could make this step less precise and result in both false positives and false negatives.
Halfond et al [24] presented and evaluated a new tool AMNESIA (Analysis and Monitoring for Neutralizing SQL-Injection Attacks), that uses a model based approach designed to detect illegal queries before they are executed on the database. The tool is the combination of static analysis and runtime monitoring. In its static part, the technique uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, the technique uses runtime monitoring to inspect the dynamically generated queries and check them against the statically built model.

Buehrer et al [13] proposed a technique based on manipulation and hence eliminates SQL Injection Vulnerabilities. At runtime the parse tree of SQL statement shows that the before inclusion of user input is compared with after inclusion of user input. In this approach, the model is expressed as a grammar that only accepts legal queries. SQLGuard is a single static class, which provides parsing and string building capabilities. This is used to dynamically generate, through concatenation, a string representing an SQL statement and incorporating user input. In this technique, the developer has to either rewrite code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query.

Su et al [69] presented the first formal definition of command injection attacks in the context of web applications; this work gives an algorithm for preventing them based on context-free grammars and compiler parsing techniques. SQLCheck identifies SQLIA’s by using an augmented grammar and distinguishing untrusted inputs from the rest of the strings by means of a marking mechanism. A guarantee of completeness of usage for SQLCHECK implies that SQL Command Injection Attacks (SQLCIA’s) will not occur. The main weakness of this approach is that it requires the manual intervention of the developer to identify and annotate untrusted sources of input, which introduces incompleteness problems and lead to false negatives.
Pietraszek and Berghe [59] introduced CSSE (Context-Sensitive String Evaluation), a method to detect and prevent injection attacks. It modifies the PHP interpreter to track precise taint information about user input. If untrusted input has been used to create certain types of SQL tokens then techniques are context sensitive analysis to detect and reject queries. CSSE requires neither application developer interaction nor application source code modifications. Since only changes to the underlying platform are needed, this method is effective against most types of injection attacks; the disadvantage of this method is less error-prone than other solutions.

McClure [49] proposed SQL DOM (Document Object Model) the uses of encapsulation for database queries to provide a safe and reliable way to access databases. These techniques offer an effective way to avoid the SQLIA problem by changing the query building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API. (SQL DOM can be considered instances of defensive coding.) Although effective, these techniques have the drawback that they require developers to learn and use a new programming paradigm or query-development process, SQL DOM automatically from an existing database schema demonstrate its applicability to solve the mentioned problems and evaluate its performance.

William R. Cook [17] described Safe Query Objects a technique for representing queries as statically typed objects and supporting remote execution by a database server. Safe query objects use object-relational mapping and reflective Meta programming to translate query classes into traditional database queries. The model supports complex queries with joins, parameters and dynamic criteria. Safe Query Objects can be considered instances of defensive coding. Although these techniques have the drawback that they require developers to learn and use a new programming paradigm or query-development processed.

Valeur et al [74] proposed the use of an Intrusion Detection System (IDS) to detect SQLIA. Their IDS system is based on a machine learning technique that
is trained using a set of typical application queries. The technique builds models of the typical Queries and then monitors the application at runtime to identify queries that do not match the model. This technique based on learning, can generate large number of false positive in the absence of an optimal training set.

Stephen W. Boyd [12] proposed a system SQL rand and provides a framework that allows developers to create SQL queries using randomized keywords instead of the normal SQL keywords. A proxy (intermediate layer) between the web application and the database intercepts SQL queries and derandomizes the keywords. The SQL keywords injected by an attacker would not have been constructed by the randomized keywords and thus the injected commands would result in a syntactically incorrect query. Since SQL rand uses a secret key to modify the keywords its security relies on attackers not being able to discover this key. The main disadvantage is that the SQL rand tool requires the application developer to rewrite code.

Konstantinos Kemalis [37] proposed a prototype SQL injection detection system (SQLIDS), it is a novel specification-based methodology for the detection and the exploitations of SQL Injection Vulnerabilities. This system monitors Java-based applications and detects SQL Injection Attacks in real time. This approach on the one hand utilizes specifications that define the intended syntactic structure of SQL queries that are produced and executed by the web application.

Nuno Antunes et al [55] demonstrate that it is possible to develop a vulnerability scanner for web services that performs much better than the commercial ones currently available. This approach is used to detect most common and most critical types of SQL vulnerabilities in web environments. This approach performs much better than well-known commercial tools, achieving very high detection coverage while maintaining the false positives rate quite low.

Debasish Das [19] proposed an effective matching mechanism detection method DUD dependent on a user defined threshold for the SQL injection based
on dynamic query matching. An attempt has been made to classify the SQL Injection Attacks based on the vulnerabilities in web applications. A brief review of the existing approaches for the detection of SQL Injection Attack also has been presented. The mechanism used in DUD is fast and effective due to it is easy to implement matching logic, SQL master file is adaptively updated No restriction is imposed on user input, strings/characters. The DUD approach is independent of the developer’s initialization of syntactical rules. Also, DUD is significant in view of its simple detection mechanism as well as its high detection rate.

Massimo Ficco [48] proposed an effective approach for improving detection performance. This is used to reduce both the rate of false positives and the percentage of undetected intrusions. This approach is used to increase the detection coverage; the process is used to analyze and correlate the symptoms which indicate anomalous and suspicious activity, and provides comments on the preliminary results.

In information security tasks are all related to managing and reducing the risk related to usage of sensitive data’s in an organization through web applications. Information security is not to reduce vulnerability in a web application. Security is a component of a web application and it is set for coordinated activities to detect and prevent the malicious activities [31].

Cristian Pinzón [18] presented an agent specialized in the detection of SQL Injection Attacks. The agent incorporates a Case-Based Reasoning Engine which is equipped with a learning and adaptation capacity for the classification of malicious codes. The agent also incorporates advanced algorithms in the reasoning cycle stages. The reuse phase uses an innovative classification model based on a mixture of a neuronal network together with a Support Vector Machine (SVM) in order to classify the received SQL queries in the most reliable way. Finally, a visualisation neural technique is incorporated, which notably eases the revision stage carried out by human experts in the case of suspicious queries.
Classifier agent was tested in a real-traffic case study and its experimental results, which validate the performance of the proposed approach, are presented here.

Sun [70] proposed an effective approach for detecting and preventing SQL Injection Attacks. Author claims that the proposed technique satisfy the following points (1) It is resistant to evasion techniques, such as hexadecimal encoding or inline comment (2) It does not require analysis or modification of the application source code (3) It does not need training traces (4) It does not require modification of the runtime environment, such as PHP interpreter or JVM (5) It is independent of the back-end database used.

Laranjeiro et al [42] proposed an approach to secure web services against SQL and XPath Injection attacks, by transparently detecting and aborting service invocations that try to take advantage of potential vulnerabilities. This mechanism applies to secure several web services specified by the TPC-App benchmark.

2.6 INTRUSION DETECTION TECHNIQUES

Fonseca et al [32] proposed an intrusion detection mechanism for the detection of malicious transactions in DBMS. This approach consists of a complete representation of user database utilization profiles to perform concurrent intrusion detection. The proposed mechanism is generic and can be easily implemented in commercial and open-source DBMS. It uses a representation of the profile of the user utilization of the database using three levels of detail (command, transaction and session) to detect the intrusions.

Maor [44] demonstrated various SQL injection signature evasion techniques. Author concludes that signature protections alone are not a practical defense against SQL Injection. A reasonably sized signature database does not provide reliable protection while a comprehensive signature database results in excessive management overhead, dramatic performance limitations and false positives. Recently, better detection strategies like SQLIA signature detection
have been proposed by IDS/IPS vendors but their success is still limited to a small subset of the whole range of attack mechanisms.

Christopher Kruegel [15] proposed an intrusion detection system that uses a number of different anomaly detection techniques to detect attacks against web servers and web-based applications. The system analyzes the user input and refers the serverside programs, the use of application-specific characterization of the invocation parameters allows the system to perform focused analysis and results a less number of false positives. The system derives automatically the parameter profiles associated with web applications (e.g., length and structure of parameters) and relationships between queries (e.g., access times and sequences) from the analyzed data. Therefore, this mechanism can be deployed in different application environments without having to perform time-consuming tuning and configuration.

Kruegel [39] proposed an anomaly based approach which utilized six different models to detect web attacks in HTTP requests that contain a query section. This paper proposed a character distribution model. The model utilized the chi-square test applied to character frequencies arranged in six groups. The authors of that work make the observation that the character distribution model did not prove to be very useful in the detection of input validation attacks. This paper, utilizes a more fine grained model which is capable of accurately detecting SQL Injection Attacks.

Shuo Dai [83] proposed Device Driver Kit (DDK) a defensive mechanism for dynamic web security. This work makes an in-depth look at the pervasive dynamic security with the interaction of IDS and firewall. Using the NDIS Network Driver Interface Specification (NDIS) intermediate driver to realize non-by passing monitoring of all packets. Rational use of protection system will help improve the server's capability to resist hackers and realize security.
Marco Cova [46] proposed Swaddler, a novel approach to the anomaly-based detection of attacks against web applications. Swaddler analyzes the internal state of a web application and learns the relationships between the application’s critical execution points and the application’s internal state. Swaddler is able to identify attacks that attempt to bring an application in an inconsistent, anomalous state, such as violations of the intended workflow of a web application.

Rietta [61] examined the threat from SQL Injection Attacks, the reasons traditional database access control is not sufficient to stop them and some of the techniques used to detect them. Moreover, the proposed model is for an anomalous SQL detector which observes the database traffic from the perspective of the database server itself. The proposed anomaly model can be used in conjunction with the existing methods to give the database server a way to mitigate the SQL injection risk that is a major application security problem.

Anomaly-based network intrusion detection systems can take into consideration packet headers, the payload or a combination of both. Damiano Bolzoni [11] argues that payload-based approaches are becoming the most effective methods to detect attacks. Nowadays, attacks aim mainly to exploit vulnerabilities at application level. The payload contains the most important information to differentiate normal traffic from anomalous activity.

2.7 CRYPTOGRAPHIC APPROACHES

Shaukat Ali et al [64] proposed a technique using hash values of username and password to improve the authentication process. Shaukat Ali et al built a prototype, SQL Injection Protector for Authentication (SQLIPA), for the evaluation of proposed idea. This is the first method to prevent SQLIA by using cryptographic hash functions.

2.8 GENERAL TECHNIQUES AGAINST SQLIA’S

Engler and Ashcraft [9] proposed metacomilation (MC) an automated checker that identifies input validation problems. MC is not directly applicable in
this context because it primarily focuses on bounds checking for integers rather than string values, it assumes that any type of input sanitizing function is sufficient to ensure that an input value is not tainted whereas often input sanitizing is performed it is difficult to specify general, application-independent validation rules.

Halfond et al [25] presented an extensive review for the different types of SQL Injection Attacks known up to date. This paper presented the techniques of various attack and shows how the attacks gain information from web application. This work shows the intent of the attacker and Injection mechanism of SQL Injection. This review classified and analyzed existing detection and prevention techniques against SQL Injection Attacks. They provide strengths and weaknesses for each technique in addressing the entire range of SQL Injection Attacks.

Kiezun et al [1] presented a technique for finding security vulnerabilities in Web applications. SQL Injection (SQLI) and Cross-Site Scripting (XSS) attacks are widespread forms of attack in which the attacker crafts the input to the application to access or modify user data and execute malicious code. The technique generates sample inputs and mutates the inputs to produce concrete exploits. The first analysis of this approach is an awareness that precisely addresses second-order XSS attacks. This technique creates real attack vectors, has few false positives and incurs no runtime overhead for the deployed application. It works without requiring modification of application code and handles dynamic programming-language constructs.

Balzarotti [10] proposed Saner which combines static and dynamic analyses to find potential XSS and SQLI vulnerabilities. Saner focuses on the sanitization process and abstracts away other details of the application, i.e., Saner creates attack vectors only for extracted, possibly infeasible, paths from the static dependency graph Saner also reports vulnerability whenever a path from source to sink contains no custom sanitation. The path may be infeasible or not exploitable. Saner tests each source-to-sink path independently and may miss attacks in which
output is constructed from multiple sinks. To detect attacks, Saner simply searches for specific strings in the output.

Recent frameworks for web applications provide a functionality that can be used to prevent SQL Injections. For example, Struts [67] supports a validator. A validator verifies that an input from the user conforms to the pre-defined format of each parameter. If a validator prohibits an input from including meta-characters, SQL Injections can be avoided.

Lam et al [40] combined static analysis, model checking and dynamic monitoring. Lam et al described a language called PQL (Program Query Language) that allows users to declare and specify information flow patterns succinctly and declaratively. This method developed a static context-sensitive, but flow-insensitive information flow tracking analysis that can be used to find all the vulnerabilities in a program. Here in web application security information flow is directly and immediately applicable.

Wagner et al [76] evaluated three tools (FindBugs, PMD and QJ Pro) and compared their efficiency with the effectiveness of code reviews. The tools achieved higher efficiency than the reviews in detecting software bugs in five industrial Java-based applications and author claims that all the tools presented false positive rates higher than thirty percentages. This evaluation did not consider security issues.

Romil Rawat [62] has proposed coding flaws at different platforms and their solutions. This concept provides a secure application based on secure guidelines. Here application loopholes are closed by secure checkpoints. Web application is vulnerable only, if there is designing flaws, coding mistakes and lack of proper guidelines. This system used advanced guidelines, which shows some rules which should be strictly followed by designer and coder. Some old methods which are vulnerable to attack should be strictly banned. Expert ways are used to eliminate any possible attack scenario and its terror.
Marco Cova et al [45] proposed a Swaddler, which analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state. This work describes an anomaly based approach which utilizes the character distribution of certain sections of HTTP requests to detect previously unseen SQL Injection Attacks. This approach requires no user interaction and no modification either the backend database or the source code of the web application itself.

Sushila Madan [72] developed a threat risk model ADMIRE which is a complete, structured and stepwise approach, which is used to identify and mitigate Code Injection Attacks and shield the database lying in the database server. Every dynamic query must be protected. It is obvious that safeguarding of security is becoming more difficult because the possible attack technologies are becoming increasingly sophisticated. Software rooted vulnerabilities like SQL Injection can be prevented, if the developers seriously incorporate the rule of validation while developing web applications.

Merlo [53] presented an investigation of the evolution of security vulnerabilities as detected by propagating and combining granted authorization levels along an inter-procedural Control Flow Graph (CFG) together with required security levels for database accesses with respect to SQL-Injection Attacks.

Nuno Antunes [56] proposed an approach to benchmark the effectiveness of vulnerability detection tools in web services. This is used to define a concrete benchmark targeting tools able to detect SQL Injection Vulnerabilities. Several tools have been benchmarked including commercial and open-source tools.

Yu-Chin Cheng [84] proposed a type of novel Embedded Markov Model (EMM) to detect different web application attacks, monitor the on-line user behavior and defend the malevolent user promptly. EMM approach detects user’s invalidated input errors and also finds out the unreasonable page transition behavior. This approach is very easy to defend the malevolent or silly user
behavior to avoid the further web system failures and sensitive information disclosure. EMM method can discover the abnormal behavior of malevolent user and detect the invalidated input attacks like SQL injection, XSS and string buffer overflow attacks.

Yang Haixia [82] presented a scheme of database security testing to detect potential input points of SQL Injection, automatically generate test cases and find vulnerability of databases by running these test cases to make a simulation attack to an application. The database security testing can stop up SQL attack channels in the beginning. This scheme proposed in this paper can detect SQL Injection Points relatively correctly and quickly, but its detection ability should be improved. The attack rule library should be extended and improved.

Anderson [3] and Stallings [65] refer to vulnerabilities as breeches in security mechanisms that can be used to perform attacks and thus constitute a threat to a computer system. Web application vulnerabilities are the main causes of any kind of attack [41] like SQL Injection Attack, Denial of service attack, Authentication and authorization based attacks, etc.

In 2007, a number of creative and lethal attacks are identified in web applications. Web site hacking continued to gain momentum as hackers had a field day exploiting vulnerabilities across all geographies and across different types of Web applications. From SQL Injection Robot to a Russian Malware gang attacking a government site [57] to exploitation of various Google vulnerabilities to various universities [16] attacks continues. Financial gains continue to be the primary goal of web based attacks. These kinds of attacks can also be used to steal intellectual property, student records, and to deface Web sites.

Alonso et al [2] described a deep analysis of the LDAP (Lightweight Directory Access Protocol) Injection technique and a clear distinction between classic and blind injection techniques presented. LDAP services facilitate access to networks information organizing it in a hierarchical database that allows
authorized users and applications to find information related to people, resources and applications. It is an essential service in almost all networks. LDAP Injection techniques are an important threat for these environments, specially, for the control access and privileges and resources management. These attacks modify the correct LDAP queries, altering their behavior for the attacker benefit and the consequences of these attacks can be very severe. Author prescribed as a unique in providing a rigorous analysis of LDAP Injection techniques and in showing representative examples of the possible effects of these attacks.