Classification of SQL Injection Attacks

San-Tsai Sun, Ting Han Wei, Stephen Liu, Sheung Lau
Electrical and Computer Engineering, University of British Columbia
{santsais,tinghanw,stephenl,sheungl}@ece.ubc.ca

***Abstract*—Most web applications deployed today are vulnerable to SQL injection attacks. The reason for this pervasiveness is that web applications and detection systems do not know the attacks thoroughly and use limited sets of attack patterns during evaluation. To address this problem, this paper presents a semantic-aware, easy to comprehend SQL injection attack model and classification scheme. The proposed classification covers aspects of SQL injection attacks that are not included in existing classification schemes. To evaluate the classification scheme, we build an attack repository by collecting SQL injection attacks from various internet sources. This classification and repository can be used to help developers and administrators understand better SQL injection attacks and evaluate defending mechanism more thoroughly.**

Last Modification Date: 2007/11/17

Revision: #14

***Index Terms*—SQL Injection Attacks, Classification, Web Application Security, Intrusion Detection**

I. Introduction

SQL injection is a class of code-injection attacks in which input data provided by a user is included in a dynamically constructed SQL query and treated as SQL code[1]. For database-reliant web sites, SQL injection vulnerabilities are frequently exploited by attackers since they are easy to find and penetrate[2], [3], [4], [5], [6]. A study conducted in 2005 by the Gartner Group found that on over 300 tested web sites, 97% were vulnerable to SQL injection attacks[7], [1]. Upon detecting SQL injection vulnerabilities, attackers commonly extract or modify data from the compromised web site. However the threats posed by attackers go beyond simple data manipulation[2], [3], [4]. Through SQL injection attacks, an attacker may extract undisclosed data, bypass authentication, escalate privileges, modify the content of the database, execute a denial-of-service attack, or execute remote commands to transfer and install software[2], [3], [4].

The root cause of such prevalent SQL injection vulnerabilities is that web applications and intrusion detection systems use only limited set of attack patterns for evaluation[8], [9], [10], [11]. Sophisticated attacks that employ evasion techniques can easily circumvent most of the detection mechanism employed today[8]. In addition, even if the injected code is intercepted before execution, administrators are often presented with information that does not identify clearly the association between the commands that were attempted, the assets that were at risk, the threats that were imposed, and the countermeasures he/she has at disposal.

To address these issues, a repository of SQL injection attacks that are classified in a semantic-aware, easy to comprehend model is needed. Previous efforts have focused on classifying web-based attacks in general, of which SQL injection is classified as a special form of code injection attack [12]. One classification that focuses primarily on SQL injection attacks has been conducted by Halfond, Viegas, and Orso [1]. In that paper, the attacks are classified according to injection mechanism, attack intent, and the attack type. However, many aspects of SQL injection attacks, such as evasion techniques, result retrieval techniques, assets, threats, vulnerabilities, the DBMS in question, and countermeasures, are not included. Moreover, semantic relations between the categories are not established, making the classification difficult to comprehend in a broad perspective.

This paper presents an improvement on existing classifications by proposing a new classification model that has the properties of being mutually exclusive, exhaustive, unambiguous, repeatable, accepted, and useful[13]. To evaluate the classification scheme, we build an attack repository by collecting SQL injection attacks from white papers, technical reports, web advisories, hacker on-line communities, web sites, and mailing lists. The proposed classification model is used to organize the entries within the repository of SQL injection attacks, of which two entries will be detailed as examples in this paper. This repository of SQL injection attacks can be used to help programmers and designers understand SQL injection attacks more thoroughly. It can also be used in the evaluation of defensive coding practices and intrusion detection systems. Also, the classification scheme is constructed in such way that security professionals or server administrators are able to identify which attacks may be particularly relevant to them, and conduct an automated evaluation of their server's security.

The rest of the paper is organized as follows. In Section II we present a SQL injection attack model. Our proposed classification scheme is discussed in Section III. Examples of applying the classification scheme are given in Section IV. The evaluation of the classification model will be found in Section V. Finally, we conclude and suggest future possible applications in Section VI.

II. SQL Injection Attack Model

An SQL injection attack has a set of properties, such as assets under threat, vulnerabilities being exploited and attack techniques utilized by threat agents. We introduce a model that represents properties associated with an attack and relationships between those properties. Figure 1 illustrates the SQL injection attack model. The semantic representation of a SQL injection attack model is as following:

- Threat agents attempt to gain access to particular *assets*

and impose particular *threats* by utilizing particular *attacks techniques*.
- The attack targets particular *DBMS* with particular *intentions* by exploiting particular *vulnerabilities*.
- The attack employs particular *evasion techniques* in order to evade detection.
- The owners of assets could deploy particular *countermeasures* to eliminate vulnerabilities.

An SQL injection attack described using the aforementioned attack model can help developers and administrators to better understand attacks and build more secured applications. The SQL injection attack model serves as the blueprint of our classification.

III. Classification

The detail feature set of every property in the SQL injection attack model is identified in this section.

A. Attack Intention

When a threat agent utilizes a crafted malicious SQL input to launch an attack, the attack intention is the goal that the threat agent tries to achieve once the attack has been successfully executed.

- *Identifying Injectable Parameters*:
  Injectable parameters are input data within a HTTP request which are directly used by server-side program logic to construct SQL statement without sufficient input validation. In order to launch an successful attack, a threat agent must first discover which parameters within a HTTP request of a specific URL are vulnerable to SQL injection attack.
- *Identifying Database Finger-Print*:
  Database finger-print is the information that identifies a specific type and version of database system. Every database system employs a different proprietary SQL language dialect. For example, the SQL language employed by Microsoft SQL server is T-SQL while Oracle SQL server uses PL/SQL . In order for an attack to be succeeded, the attacker must first find out the type of and version of database deployed by a web application, and then craft malicious SQL input accordingly.
- *Discovering Database Schema*:
  Database schema is the structure of a database system. The schema defines the tables, the fields in each table, and the relationships between fields and tables. Database schema is used by threat agents to compose a correct subsequent attack in order to extract or modify data from database.
- *Bypassing Authentication*:
  Authentication is a mechanism employed by web application to assert whether a user is who he/she claimed to be. Matching a user name and a password stored in the database is the most common authentication mechanism for web applications. Bypassing authentication enables an attacker to impersonate another application user to gain un-authorized access.
- *Extracting Database Data*:
  Database data used by a web application could be sensitive and highly desirable to threat agents. Attacks with intention of extracting data are the most common type of SQL injection attacks.
- *Modifying Database Data*:
  Database data modification provides a variety of gains for a threat agent. For instance, a hacker can pay much less for a online purchase by altering the price of a product in the database. Or, the threads in a online discussion forum can be modified by an attacker to launch subsequent Cross-Site-Scripting attacks.
- *Downloading File* :
  Downloading files from a compromised database server enable an attacker to view file content stored on the server. If the target web application resides on the same host, sensitive data such as configuration information and source code will be disclosed too.
- *Uploading File* :
  Uploading files to a compromised database server enable an attacker to store any malicious code onto the server. The malicious code could be a Trojan, a back door or a worm that can be used by an attacker to launch subsequence attack.
- *Executing Remote Commands*:
  Remote commands are executable code resident on the compromised database server. Remote command execution allows an attacker to run arbitrary programs on the server. Attacks with this type of intention could cause entire internal networks being compromised.
- *Escalating Privilege* :
  Privileges are described in a set of rights or permissions associated with users. Privilege escalation allows an attacker to gain un-authorized access to a particular asset by associating a higher privilege set of rights with a current user or impersonate a user who has higher privilege.

B. Assets

Assets are information or data an unauthorized threat agent attempt to gain.

- *Database Server Fingerprint*:
  The database server fingerprint contains information about the database system in use. It identifies the specific type and version of the database, as well as the corresponding SQL language dialect. A compromise of this asset may allow attackers to construct malicious code specifically for the SQL language dialect in question.
- *Database Schema*:
  The database schema describes the server's internal architecture. Database structure information such as table names, size, and relationships are defined in the database schema. Keeping this asset private is essential in keeping the confidentiality and integrity of the database data. A compromise in the database schema may allow attackers to know the exact structure of the database, including table, row, and column headings.
- *Database Data*:
  The database data is the most crucial asset in any database system. It contains the information in the tables described

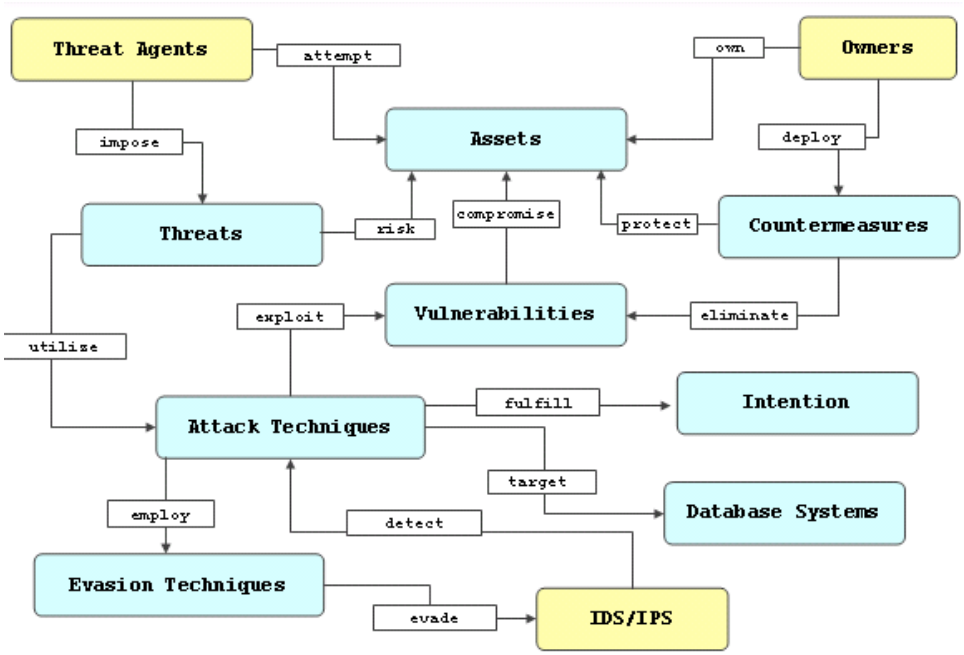

Fig. 1. SQL injection attack model.

in the database schema, such as prices in an online store, personal information of clients, administrator passwords, etc. A compromise in the database data will usually result in failure of the system's intended functionality, thus, its confidentiality and integrity must be protected.

- *Host*:
  A host is a discrete node in any network, usually uniquely defined with an IP address. It may have various privileges in a network and may be a database server or a regular computer terminal.
- *Network*:
  A network interconnects numerous hosts together and allows communication between them. A compromise in a network will most likely compromise every host in the network. Some networks may also be interconnected with other networks, furthering the potential damage, should an attack be successful.

C. Threats

Threats are potential violation of security. There are four types of threats: disclosure, deception, disruption and usurpation.

- Disclosure: Unauthorized access to information.
- Deception: Acceptance of false data. Examples of deception are modification of data, spoofing, repudiation of origin and denial of receipt.
- Disruption: Interruption or prevention of correct operation. Examples of disruption are modification of data, and denial of service.
- Usurpation: Unauthorized control of some or all parts of the system. Examples of usurpation are modification of data, spoofing, delay of service and denial of service.

D. Vulnerabilities

- *Insufficient Input Validation*:
  Input validation is an attempt to verify or filter any given input for malicious behavior. Insufficient input validation will allow code to be executed without proper verification of its intention. Attackers taking advantage of insufficient input validation can utilize malicious code to conduct attacks.
- *Privileged Account*:
  A privileged account has a degree of freedom to do what normal accounts can not. Its actions may also be exempt from auditing and validation. This presents a vulnerability since a jeopardized privileged account, such as an administrator account, can compromise much more than what a jeopardized regular account can.
- *Extra Functionality*:
  Extra functionalities meant to provide a broader range of usage may be a vulnerability since a combination of these functionality may result in unintended actions. For example, xp_cmdshell is meant to provide users with a way of executing operating system commands, but is commonly used to added unauthorized users into the operating system.

E. Attacks Techniques

Attack techniques are the specific means by which a threat agent carries out attacks using malicious code. Threat agents may use many different methods to achieve their goals, often combining several of these sequentially or employing them in different varieties [1].

- *Tautology*:
  This technique relies on injecting statements that are always true so that queries always return results upon

evaluation of a WHERE conditional. A common example would to be inject a ”or 1=1” into the ”login” parameter.

- *End of Line Comment*:
  After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments. An example would be to add ”- -” after inputs so that remaining queries are not treated as executable code, but comments. This is useful since threat agents may not always know the syntax or fields in the server.
- *Illegal/Logically Incorrect Query*:
  This technique is usually used by the threat agent during the information gathering stage of the attack. Through injecting illegal/logically incorrect requests, an attacker may gain knowledge that aids the attack, such as finding out the injectable parameters, data types of columns within the tables, names of tables, etc.
- *Union Query*:
  Threat agents use this technique to guide servers to return data that were not intended to be returned by the developers. A common example would be to add the statement ”UNION SELECT”, along with an additional target dataset so that queries return the union of the intended dataset with the target dataset.
- *Piggy-backed Query*:
  The threat agent may add additional queries beyond the intended query, effectively ”piggy-backing” the attack on top of a legitimate request. This technique relies on server configurations that allow several different queries within a single string of code. For example, the threat agent may add a query delimiter such as ”;”, and then follow up with a command of his/her own, such as ”drop table <name>”, which effectively deletes the table specified.
- *System Stored Procedure*:
  Database server often ship with system stored procedures that programmers may use when developing application. If the threat agent has knowledge of which back-end server is running, he/she may be able to exploit these stored procedures to perpetrate their attacks. Stored procedures may yield results that go beyond the database itself, but also interact with the OS, for example.
- *Blind Injection*:
  With sufficiently secure systems, threat agents may probe for vulnerable parameters or extract data by using this technique. Blind injection allows threat agents to infer the construct of the database through evaluating expressions that are coupled with statements that always evaluate to true and statements that always evaluate to false. For example, the threat agent can add ”and 1=0 –” for one attempt, while ”and 1=1 –” is used for another attempt, both added onto the same query. Through examining the behavior of the server, the threat agent may then deduce whether the particular parameter is vulnerable or not, where the two attempts result in the same behavior, the parameter is secure, while different behavior resulting from the two statements suggest that the parameter is vulnerable.
- *OPENROWSET Result Retrieval*:
  When trying to exploit SQL injection in an application, an attacker needs a method of retrieving the results. The OPENROWSET function allows a user in SQL Server to open remote data sources. The function OPENROWSET is most commonly used to pull data into SQL Servers to be manipulated. They can however also be used to push data to a remote SQL Server. Below is an example of pushing data to an external data source:

```
insert into
OPENROWSET('SQLoledb',
'server=servername;uid=sa;pwd=HACKER',
'select * from table1')
select * from table2
```

  In the example above, all rows in table2 on the local SQL Server will be appended to table1 in the remote data source.

F. Evasion Techniques

Evasion techniques are obscuring techniques employed in an attack to avoid detection by signature-based detection systems [8]. In the context of SQL injection detection, a signature is the pattern of known attack strings. SQL injection attack occurs when input string alter the intended syntactical structure of SQL statement. Thus, an attack signature usually consist of one or more SQL keywords, deliminators and expressions. Signature-based detection systems build a database of attack signatures, and then examine input strings against the signature database at runtime in detection of attacks. Evasion techniques obscure input strings, making look different but yielding the same results when executed by a database server.

- *Sophisticated Matches*: One of the common signatures used by such mechanisms is some sort of variant on the famous OR 1=1 attack. Sophisticated matches evasion technique uses alternative expression of *”OR 1=1”*. For example: *OR ’Unusual’ = ’Unusual’* , *OR ’Simple’ = ’Sim’+’ple’*, *OR 2 > 1* and *OR ’Simple’ BETWEEN ’R’ AND ’T’* all have the same effect as *”OR 1=1”*.
- *Hex Encoding*
  Hex encoding evasion technique uses hexadecimal encoding to represent a string. For example, the string ’SELECT’ can be represented by the hexadecimal number 0x73656c656374, which most likely will not be detected by a signature protection mechanism.
- *Char Encoding*:
  Char encoding evasion technique uses build-in CHAR function to represent a character. For example, the string ’SELECT’ can be represented by the CHAR function as char(73)+char(65)+”LECT”, which make it very difficult for detection system to build a signature that match it.
- *In-line Comment*:
  In-line comment evasion technique obscures input strings by inserting in-line comments between SQL keywords. For instance, */**/UNION/**/SELECT/**/name* can escape detection from signatures that expects white space between SQL keywords.
- *Dropping White Space*:
  Dropping white space evasion technique obscures input strings by dropping white space between SQL

keyword and string or number literals. For example, *OR'Simple'='Simple'* works exactly the same way as *OR 'Simple' = 'Simple'*, but has no spaces in it, make it capable of evading any spaces based signature.
- *Break Words in the Middle*:
  With MySQL, the in-line comments would not work as a replacement for a space. The in-line comments can be used in MySQL to break words in the middle, for instance: *UN/**/ION/**/ SE/**/LECT/**/* is evaluated as *UNION SELECT*.

G. DBMSs

Although every database management system in the market support ANSI/ISO standard Structured Query Language , each vender also develops a proprietary SQL language dialect. Almost every SQL injection attack within attacks we found target a specific database. Common targeted DBMSs are list as follows:

- MS SQL Server
- MySQL
- Oracle
- DB2
- Sybase
- Informix

H. Countermeasures

There are a number of ways a programmer/system administrator can prevent or counter attacks made on their systems.

- *Parameterized Query*:
  Parameterized query is parameterized database access API provided by development platform such as PrepareStatement in Java or SQLParameter .NET. Instead of composing SQL by concatenating string , each parameter in a SQL query is declared using place holder and input is provided separately.
- *Least Privilege*:
  The account that an application uses to access the database should have only the minimum permissions necessary to access the objects that it needs to use.
- *Different Accounts*:
  Use a different database account for a task that requires a different level of privilege.
- *Customized Error Message*:
  Threat agents may gain access to knowledge through overly informative error messages, yet completely removing error messages makes debugging a difficult task. Customized error messages hinder the reconnaissance progress of threat agents, particularly in deducing specific details such as injectable parameters, etc.
- *System Stored Procedure Reduction*:
  Once a threat agent gains knowledge of which back-end server is used, he/she has knowledge of an entire set of system stored procedures that are available. By limiting the system stored procedures one can execute on a server, especially the processes that are not used, one can reduce or even eliminate vulnerabilities that may arise from these stored procedures.
- *SQL Keyword Escaping*:
  Escape specific SQL keyword or deliminator in the input string.
- *Input Variable Length Checking*:
  By checking for input variable length, malicious code strings beyond certain length limits will not be applicable. Even if the length limitation is long enough to fit a few additional queries, the inability to input an infinitely long string disables the threat agent from employing evasion techniques such as encoding, and consequently, allows signature based detection mechanisms to intercept simple attacks.

IV. Examples of Attack Classification

This section illustrates how the classification scheme discussed in the previous section can be used to categorize an SQL injection attack.

A. Example 1

As the first example, let us consider the following attack:

```
' UNION SELECT '_HACKER',TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES --
```

The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the list of all table names in the database. The above attack string can be categorized as following:

- Threat agents attempt to gain *database schema* assets on the database host.
- Threat agents impose *disclosure* threats on the asset and exploit *insufficient input validation* vulnerability of the web application.
- The attack utilizes *end of line comment*, and *union query* attacks techniques.
- The intention of the attack is to *discover database schema*.
- There are no evasion techniques used in order to evade detection.
- The DBMS that is vulnerable to this attack is *MS SQL Server* and *Sybase*.
- The owner of asset could deploy *parameterized query*, *SQL keyword escaping* and *input variable length checking* countermeasures to eliminate vulnerabilities.

B. Example 2

As the second example, let us consider the following attack:

```
/* */declare/* */@x/* */as/* */varchar
(4000)/* */set/* */@x=convert(varchar
(4000),0x6578656320206D61737465722E2E
78705F636D647368656C6C20276E657420757
36572206861636B6572202F6164642027)/*
*/exec/* */(@x)
```

The above attack uses hexadecimal encoding and in-line comment evasion technique to obscure following attack string:

```
exec master..xp_cmdshell 'net user hacker
```

```
1234 /add
```

Once the injected code has been executed by database server, this attack adds a new user named "hacker" with the password "1234" to the operating system. This attack string can be categorized as following:

- Threat agents attempt to gain access to *host* and *internal network*.
- Threat agents impose *deception* and *usurpation* threats on the assets, and exploit *insufficient input validation* and *privileged account* vulnerabilities of the web application.
- The attack utilizes *end of line comment*, *piggy-backed query*, and *system stored procedure* attacks techniques.
- The intention of this attack is *privilege escalation*.
- The threat agent employs *dropping white space*, *in-line comment* and *hexadecimal encoding* evasion techniques in order to evade detection.
- DBMS vulnerable to this attack is *MS SQL Server*.
- The owner of asset could deploy *parameterized query*, *different accounts*, *least privilege*, *system stored procedure limitation*, *SQL keyword escaping*, and *input variable length checking* countermeasures to eliminate vulnerabilities.

V. Evaluation

The space of real attacks is unlimited. However, a new attack is usually a variation of an existing type of attack. In order to quantitatively test how large a fraction of attacks the classification covers, we build a repository by collecting SQL injection attacks from white papers, technical reports, web advisories, hacker on-line communities, web sites, and mailing lists. Entries within the repository of SQL injection attacks are categorized based on the proposed classification scheme. The result shows the classification is unambiguous: clear and precise so that classification is not uncertain, regardless of who is classifying. The classification is also repeatable: repeated applications result in the same classification, regardless of who is classifying.

VI. Conclusion

SQL injection attacks are prominent in today's web application as shown in the Gartner Group study[7], [1]. By taking advantage of the server's vulnerabilities, an attacker may find themselves able to freely edit an online store's prices, extracting personal data from a corporate database, or simply deleting the database and shutting down the network. This paper presented a new classification model in regards to SQL injection attacks, with properties of being mutually exclusive, exhaustive, unambiguous, repeatable, and useful [13].

By allowing programmers and system administrators to understand the attacks more thoroughly, more attacks will be detected and more countermeasures will be introduced into the systems. The proposed model defines attacks by means of behavior, instead of signature, in order to circumvent various evasion techniques used by the attacker. By splitting the classification into the aforementioned categories, system administrators can clearly see the exact intentions of an attack, how and what it attacks, and most importantly, how to implement countermeasures.

As seen in the examples, this model is easy to use, yet gives a complete description of the attack. Results are clear and unambiguous and do not contain any uncertainty. Classification selections are specific, do not overlap, and are exhaustive in certain categories. In the event of a new attack or evasion technique, the user can easily add an entry to the corresponding class.

Future attempts should focus on maintaining a complete and up-to-date repository of known SQL injection attacks. This ensures that an attack on one database server will result in a prevention on the other. Knowledge and awareness of SQL injection attacks is also essential. Repositories should be made public, along with voluntary reports of attacks.

References

[1] J. V. William G.J. Halfond and A. Orso, "A classification of sql injection attacks and countermeasures," 2006.

[2] C. Anley, "Advanced sql injection in sql server application," *Technical report, NGSSoftware Insight Security Research (NISR)*, 2002. [Online]. Available: http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[3] C. Cerrudo, "Manipulating microsoft sql server using sql injection," *Technical report, Application Security, Inc.*, 2003. [Online]. Available: http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf

[4] C. Anley, "(more)advanced sql injection in sql server application," *Technical report, NGSSoftware Insight Security Research (NISR)*, 2002. [Online]. Available: http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

[5] K. Spett, "Sql injection: Are your web applications vulnerable?" *Technical report, SPI Dynamics, Inc.*, 2005. [Online]. Available: http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

[6] ——, "Blind sql injection: Are your web applications vulnerable?" *Technical report, SPI Dynamics, Inc.*, 2005. [Online]. Available: http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf

[7] W. G. Halfond and A. Orso, "Amnesia: Analysis and monitoring for neutralizing sqlinjection attacks," in *20th IEEE/ACM International Conference on Automated Software Engineering.*, Long Beach, California, USA, 2005, p. 174.

[8] A. S. Ofer Maor, "Sql injection signatures evasion," *White Paper, Imperva Inc.*, 2005. [Online]. Available: http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html

[9] D. Litchfield, "Data-mining with sql injection and inference," *Technique Report, An NGSSoftware Insight Security Research (NISR) Publication*, 2005. [Online]. Available: http://www.ngssoftware.com/research/papers/sqlinference.pdf

[10] S. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in *American Conference on Neutron Scattering*, College Park, Maryland, USA, 6-10 June 2004, pp. 202–302.

[11] B. W. W. G. T. Buehrer and P. A. G. Sivilotti, "Using parse tree validation to prevent sql injection attacks," in *International Workshop on Software Engineering and Middleware*, Lisbon, Portugal, September 2005.

[12] "A new taxonomy of web attacks suitable for efficient encoding," *Computers & Security*, vol. 22, no. 5, pp. 435–449, 2003.

[13] E. Amoroso, *Fundamentals of Computer Security Technology*. Prentice-Hall PTR, 1994.