Fall 2010

Intrusion Detection And Prevention System: SQL-Injection Attacks

Varian Luong
San Jose State University

Intrusion Detection And Prevention System:

SQL-Injection Attacks

2010

A Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Varian Luong

ABSTRACT

Personally identifiable information (PII) is information regarding things such as bank accounts, retirement or stock investment accounts, credit card accounts, medical records, or insurance claims. There is a need to protect the PII in databases that are connected to the ubiquitous, global network that is the Internet. If there is any vulnerability in the protection in a system that holds PII, then it presents an opportunity for an unauthorized person to access this PII. One of the techniques available to would-be information thieves is SQL injection (SQL-I). In this project, a system is developed to analyze the values submitted by users through HTML forms and look for possible attack patterns. Once the system finds such a pattern, it blocks the attack and makes a record of the activity. If an attacker continues to pass such attack patterns, the system blocks access by this user altogether. A mechanism is included to block users who attempt to log in at an abnormally high rate. This provides a combination of pattern-based detection and anomaly-based detection to create a reasonably robust intrusion detection system, with respect to SQL-I attacks.

TABLE OF CONTENTS

1. INTRODUCTION

The Internet is a huge interconnected network, the largest in the world. In less than two decades, it has grown from an esoteric academic medium to a ubiquitous source of information. A great deal has been made about the accessibility of the Internet as well as how it will supposedly change all of our lives. Computer files are replacing paper files as electronic records in institutions such as hospitals, insurance providers, and banks are quickly replacing their carbon-based counterparts. As they are getting used to the idea of using it, people are going shopping and banking over the Internet. People are demanding more and more access to the Internet – always wanting the information faster, more constantly available, and increasingly diverse in content. Insurance providers are starting to encourage employers, physicians, and insurance brokers to submit medical claims information electronically in order to cut down on costs and improve turnaround time.[13] With the growth of digital traffic, we are affording ourselves great convenience, but also exposing ourselves to greater risk of having sensitive information intercepted or stolen. Systems that contain sensitive information must have safeguards to prevent the risk of external attacks on the system, for example with the use of some type of tool. We discuss the development of a software tool in this paper.

Personally identifiable information (PII) can be found in bank accounts, retirement or investment accounts, and credit card accounts. The volume of this information is often great as a result of institutions having so many customers and users. Therefore, the ideal way to store and retrieve all of this information is in a database. To satisfy people's desire to access information from this database from anywhere at

anytime, the natural network of choice is the Internet. The feature of having such convenient access is precisely how we run into security issues. The Internet was first created without security in mind, because it was not an issue at the time. It was a simple, open way of communicating and sharing ideas in the academic setting. The modern version of the Internet, however, is accessible by anybody including those with dubious moral values. There needs to be protection of the PII in these databases. People want their names, addresses, phone numbers, credit card numbers, and social security numbers private and protected. Now, more than ever, we need to strengthen the security of the systems which use these databases and make them secure.

One of the techniques available to the would-be information thieves is SQL-Injection (SQL-I). SQL-I attacks involve a variety of methods, but the intention of an attacker using it is to submit specially-chosen patterns when asked for the user's username and password on an internet form. The values passed from a form may be directly concatenated into a string with an SQL query string. Then the resulting SQL query string is dynamically parsed in order to test, for example, if the username and password are correct. When the text from the user input is unchecked and incorporated into the SQL query, these abnormal values can result in highly abnormal behavior such as gaining access to the system despite not supplying the correct password, selecting columns from completely different tables than the ones being queried in the original query, or even highly dangerous behavior such as deleting any tables. It is therefore imperative that these values be checked before being submitted to a database management system (DBMS) parser to be run on the database.

There are systems that have been developed to detect SQL-I vulnerabilities such as Sania. Sania identifies vulnerabilities during development and debugging phases by examining the SQL queries that occur between a web application and the database, and generating elaborate attacks according the vulnerabilities it finds.[10] There has also been work on tools such as AMNESIA which is a runtime SQL-I detection tool and JCrasher which generates test cases, both of which are based in the Java language.[11]

Aside from these two tools, there are many other tools both academic and commercial, such as CodeScan Labs: SQL-Injection, that identify SQL-I attack vulnerabilities.[5] However, this paper will focus on detecting and blocking SQL-I attacks at these vulnerable sites.

There is work done on using parse trees as a way of dynamically checking queries for the presence of SQL-I attacks. However, this requires the comparison between the structures of the SQL query before and after user input is incorporated into the dynamic SQL query.[12]

However, a software system can be developed to intercept the values submitted by users through HTML forms and look for these SQL-I attack patterns. This paper focuses on software approaches, although there are hardware approaches as well. Once the software system finds something that matches a SQL-I attack pattern, it makes a record of this activity. If a person continues to submit such patterns, the system blocks access of this user altogether. We also include in the system a mechanism to block users who attempt to log in at an abnormally high rate. With this combination of pattern-based detection and anomaly-based detection, we have a much more robust intrusion detection system. The attacker will be greatly slowed down and once an attack is detected and

reported, the system administrator will be given the chance to take steps to properly deal with the attack. He may choose to block the attacker's IP address, if it was not already blocked. The system administrator may examine the attacker's input and develop and add more SQL-I signatures to be used to detect similar attacks in the future.

There are also a variety of different commercial tools which deal with SQL-I attacks that are available. However, the ones studied in this paper are quite costly, or do not apply to all DBMS's, or are only useful for systems in development. The Intrusion Detection and Prevention System (IDPS) described in this paper is a free, but is both powerful and robust against SQL-I attacks on any system using any DBMS. When the IDPS is in place and these steps are followed, then the valuable information contained in the database, which may include PII information, will be properly protected.

2. SQL

SQL (pronounced "S-Q-L") is the high-level language used in numerous relational database management systems. It was originally developed in the early 1970's by Edgar F. Codd at IBM and soon became the most-widely used language for all relational databases. SQL is a declarative computer language which has elements which include clauses, expressions, predicates, queries, and statements.[1]

What makes SQL so powerful is its immense flexibility and its ability to be abstract. It allows a human being to use SQL to ask for what information he wants without outlining how the information is to be retrieved. Thus, this relieves the user of any programming knowledge needed to satisfy the query. In this sense, it is an even higher "high level" language than most traditional programming languages such as C++ and Java. A person with little to no programming background can still use SQL effectively. However, SQL does include powerful features and functions that allow users with programming knowledge to build complex queries and apply them to even more powerful uses.

2.1 Background

Enterprise-scale applications over the Internet often have a need to authenticate a large number of users, to retrieve specific information from a large warehouse of data, or to save information for later retrieval by an application or an analyst. The amount of data is so large, that it cannot be accomplished in-memory. Therefore, these Internet applications must make use of a database in the form of a database management system (DBMS). Internet applications often take data obtained from web pages and then combine them directly into an SQL statement to be directly and dynamically parsed, and

then processed by the DBMS. This allows powerful and effective queries to be written quickly and easily. In turn, these incorporated queries do jobs such as log in authentication, and data storage and retrieval.

Flexibility is another strength of SQL's. Similar to a true programming language, SQL allows the inclusion of inline comments within the code, including between statements. SQL allows pattern and regular expression matching of strings. [1] SQL also enables users to concatenate and combine separate characters or values (such as from a column or field of a record) to form a complete string. As it will be shown, it is the flexibility of SQL syntax which primarily makes signature-based detection challenging.

SQL also has a UNION construct which allows the rows from different tables to be selected simultaneously, as long as the columns are compatible types. This is a powerful feature that opens up a new dimension in table selection and enables more complicated aggregation. As with all power in the wrong hands, the UNION can be used to do things beyond the intension of the database application designers and engineers. A person may make use of all of these features and functionality in a way that is unauthorized and dangerous by using SQL-Injection attacks.

3. SQL-INJECTION

3.1 AND/OR Attack

Web programmers often take string values entered by an Internet user on a form that represents user names and passwords and place them directly into the SQL statement to be run against a database. A simple test SQL statement that may be used is the following example.

```
SELECT username, password
FROM UserAuth
WHERE username = 'usernameFromForm'
   AND password = 'passwordFromFrom';
```

In this example, the values usernameFromForm and passwordFromForm are the literal values obtained from the form. The intent is using the username and password obtained from the form to see if there is a matching username and password in the UserAuth table. If any rows are returned, the user is authenticated. However, if the web programmer is not careful and uses this method and the form values without checking them, a hacker may instead pass arbitrary values that the programmer did not originally anticipate. One such attack is the basic attack that involves the AND or OR logic in the SQL predicate. The hacker can specify a valid username such as “John Doe” and then specify the password as “' OR '1'='1” in the form. The final test SQL query that uses these values will be:

```
SELECT username, password
FROM UserAuth
WHERE username = 'John Doe'
   AND password = '' OR '1'='1';
```

Provided that “John Doe” is a valid user, the database will allow the hacker to log-in and proceed as “John Doe”, because even though the password string is not empty

(the first predicate), '1'='1' is a valid predicate that will always return TRUE. Thus, the hacker has just accessed the account without ever knowing the password, and now she has full access to the victim's information. The hacker does not need to know this is the way the form data is used to construct the SQL statement; he or she just simply needs to do several "probing" tests and see the messages returned to see if this is indeed the case. If the attack does not succeed, the attacker simply moves on and tries another method. If it succeeds, the DBMS will happily return the username and corresponding password; our hacker now has unauthorized access to the database through that username.

3.2 Comments Attack

As mentioned before, SQL allows inline commenting within the SQL "code". This allows two variations of SQL-I comments attacks. One simple variation is assigning username to be a valid username followed by comment characters. For example, we assign username = "admin' --". Then our SQL test query may look like the following.

```
SELECT username, password
FROM UserAuth
WHERE username = 'admin' --' AND password = 'anything';
```

Everything after the "--" in the WHERE clause will be ignored, so this will allow the hacker to log in as "admin"! This is a method of using comments as a way of ignoring the rest of the query.[2]

The variation of the comments attack is using comments as a way of obfuscating the signature of any SQL-I attack to avoid detection. Therefore, the use of C-style comments "/*" and "*/" can be combined with any of the previously discussed attacks as a way of attempting to circumvent signature-based detection. For example, if an application searches a string passed from a form for the UNION keyword to attempt to

catch UNION injection attacks (discussed in more detail in a subsequent section), an attacker may choose to use comments to conceal this. For example, instead of using "UNION ALL", the attacker may instead use

'UNION /**/ALL'

or

'UN/**/ION A/**/LL'.

Both of these are synonymous with "UNION ALL" in the context of an SQL statement. In addition to breaking up keywords, comments may be used in place of spaces.

A system using signature-based detection may miss keywords and SQL-I patterns if it is not careful to also consider SQL-I Comments attacks as well.

3.3 String Concatenation Attack

SQL has an option to concatenate separate strings or characters to form complete strings. This is accomplished using + or "double pipe" (||), or the function CONCAT (such as in MySQL). These operations can be used to create a variation of the UNION Injection attack by obfuscating the UNION keyword in a string concatenation operation. For example, an attacker may use

'UNI' + 'ON A' + 'LL'

in place of "UNION ALL" if he suspects the system looks for the UNION ALL keyword.[2]

Another use of string concatenation in an attack is when the attacker suspects the system searches for single quotes (‘). Then the attack may choose to use the CHAR() function in conjunction with the string concatenation to issue characters indirectly without using any single quotes. For example, an attack may use

CONCAT(CHAR(65),CHAR(68),CHAR(77),CHAR(73),CHAR(78))

to represent ‘ADMIN’ so that the system will not find a single quote if it was looking for them.

3.4 UNION Injection Attack

The UNION Injection attack may be the most dangerous, but certainly the most surprising of the SQL-I attacks. This is because if it is successful, the UNION Injection attack allows the attacker to return records from another table! For example, an attack may modify the SQL query statement that selects from the user authentication table to select another table such as the accounts table.

```
SELECT username, password FROM userAuth
UNION ALL
SELECT accountNum, balance FROM Accounts
```

The use of UNION ALL in this attack allows the attacker access to tables that the SQL query statement was not originally designed for. The resulting rows selected from both tables will appear on the resulting page.

The trickiness in this attack lies in the fact that the columns selected from the second table must be compatible in number (the same number of columns as the original table must be selected) and type. When trying to guess the correct number of columns, the attack may simply keep trying to use different number of columns in each attempt until he finds the right number. To match the type, the attack may try to try different types until he stumbles upon the right one or he may simply choose to use NULL instead. The IDPS system discussed later does not return any messages such as response or HTTP status codes and limits internal information being broadcast externally as much as possible.

3.5 Hexidecimal/Decimal/Binary Variation Attack

Attackers can further try to take advantage of the diversity of the SQL language by using hexadecimal or decimal representations of the keywords instead of the regular strings and characters of the injection text. For example, instead of using the traditional SQL-I Attack text

```
1 UNION
SELECT ALL
FROM WHERE
```

an attacker may substitute this with

```
&#x31;&#x20;&#x55;&#x4E;&#x49;&#x4F;&#x4E;&#x20;&#x53;&#x45;&#
x4C;&#x45;&#x43;&#x54;&#x20;&#x41;&#x4C;&#x4C;&#x20;&#x46;&#x5
2;&#x4F;&#x4D;&#x20;&#x57;&#x48;&#x45;&#x52;&#x45;
```

to attempt to avoid detection by signature-based detection engines.[7]

The system that does not look for hexadecimal or decimal characters will be susceptible to this variation of the SQL-I attack.

3.6 White Space Manipulation Attack

Signature-based detection is an effective way of detecting SQL-I attacks. Modern systems have the capacity to detect a varying number of white spaces around the injection code, some only detect one or more spaces; they may overlook patterns where there are no spaces in between. For example, the SQL-I pattern

```
' or 'a' <> 'b
```

can be re-written as

```
'or'a'<>'b
```

containing no spaces in between. A DBMS SQL parser will be able to handle a variable around all of white space characters or keywords. If a signature-based detection method only takes into account the first pattern, it will completely overlook the second one.

In additional to the standard space character, white space characters also include the tab, carriage return, and line feed characters.[7] To properly implement signature-based detection, the system must be able to handle white space characters.

4. EXISTING TOOLS

There are a number of existing tools available, both hardware and software based, to deal with SQL-Injection attacks. Tools exist to detect SQL-Inj attacks while others try to identify and fix SQL-Injection vulnerabilities. The following are a few software ones we will discuss.

- GreenSQL
- dotDefender
- CodeScan Labs: SQL-Injection

GreenSQL is a free Open Source database firewall that sits between the web server and the database server and is used to protect databases from SQL injection attacks. The logic is based on evaluation of SQL commands using a risk scoring matrix as well as blocking known database administrative commands (e.g., DROP, CREATE, etc). Reports are generated on timestamp, query pattern, reason blocked (e.g., true expression, has 'or' token). It has a white list of approved SQL patterns. However, only MySQL database is currently supported.[3] In comparison, the IDPS in this project may be used with any relational database, not just MySQL. The IDPS has both black and white list pattern features.

Applicure's dotDefender is a web application firewall that offers a SQL-Injection solution. dotDefender is a multi-platform solution running on Apache and IIS web servers. Central management ensures a single point of control and reporting for all servers. There is an application layer firewall in front of web applications. It has a set of security rules that enable it to be a powerful solution. However, the cost is prohibitive.

The annual license costs $1,810 while a perpetual license is $3,995, which are both pricy for personal use.[4] While dotDefender is an expensive product, IDPS is a free product.

Another product is CodeScan Labs' SQL-Injection detection product. It has the capability to scan web application source code that you selected for code syntax vulnerabilities. It subsequently generates a "debug style" report. The speed depends on how large the web application is and its complexity. The CodeScan software does not fix the code, however; it only points out the issues. The company offers a 21-day free trial, but normally it requires a yearly subscription to be maintained. The actual price is not advertised and one must contact sales representative to find out the cost. A separate activation key is required for different programming languages and additional capabilities. [5]

Table 4-1 is a summary of the comparison of the different SQL-Injection protection products described earlier.

Product	**IDPS**	**GreenSQL**	**dotDefender**	**CodeScan Labs: SQL-Injection**
How it works	Evaluates form data for known SQL-Injection attack patterns; detects CGI script attacks	Evaluates SQL commands using a risk scoring matrix	Application layer firewall in front of web applications; uses a set of security rules; uses central management for single-point control and reporting for all servers; detects SQL-I and cross-site scripting attacks	Scanner runs on and analyzes development code for cross-site scripting and SQL injection vulnerabilities
Platform	PHP	Unknown	Apache and IIS web servers	ASP with VBScript; ASP.NET C#; PHP 3, 4 and 5
Reports	Summary tables and e-mail	Yes	Unknown	Unknown
Blocks	Yes, by IP	Yes	Yes	Yes
Black and white lists	Yes	Yes	Unknown	Unknown
Databases supported	Any	MySQL	Any	Any
Cost	Free	Free	Annual license = $1,810; perpetual license = $3,995.00	Yearly subscription = price must be inquired from sales rep; need new activation key for different prog. languages, additional capabilities

Table 4-1 A comparison of different SQL-Injection Protection products

5. INTRUSION DETECTION AND PREVENTION SYSTEM (IDPS) DESIGN

5.1 Fundamentals

The system discussed is called the Intrusion Detection and Prevention System (IDPS). The particular system discussed here is an extension of a particular system that protects a web application system from CGI attacks.[6] However, the original system did not guard against SQL Injection attacks directed at databases connected to the system. We discuss how the SQL-I extension of the Intrusion Detection and Prevention System works in more detail.

5.2 IDPS Detection Models

There are two models of detection used by this system.

- Signature-based Detection Model
- Anomaly-based Detection Model

A system that implements only one of these models is not as robust as a system that utilizes a combination of them. The signature-based detection fails to detect unknown attacks, while anomaly-based detection will detect unusual activity and behavior. This is the reason why the Intrusion Detection and Prevention System (IDPS) makes use of both.

5.2.1 Signature-based (pattern) Detection Model

In the signature-based detection model, the input obtained from an HTML form is compared to known SQL-I attack patterns (or signatures). If the input is found to match a signature, access is denied and the user is given a generic invalid username/password screen. We intentionally avoid returning a page with an HTTP response or status code which will describe the error that occurred to the user. This is to limit the information and feedback the IDPS system gives to would-be intruders. Even information that seems

perfectly harmless can unwittingly give hints about how our system works to attackers which may help them to find a way to circumvent the system's protections.[8] If a user submits input that matches a known signature an arbitrary number of times, the user's IP is automatically blocked from accessing the system altogether. An administrator would have to unblock the IP in order for this user to regain access.

The signatures themselves would have to be efficient, because a database that contains too many signatures or inefficiently-written signatures would result in poor performance. Also, the signatures would have to be chosen carefully, because we would like to minimize the number of false positives returned.

The biggest flaw in the signature-based detection model is it cannot detect attacks that are unknown. That is the reason the IDPS relies on the anomaly-based detection model as a complement to the signature-based detection model.

5.2.2 Anomaly-based (behavioral) Detection Model

In the anomaly-based detection model, the number of times a user attempts to log into the system, successful or not, is considered. If the attempts from a user exceed a predetermined number, the system will lock out this user's IP for a period of time. The user may retry after this time has elapsed. It is important that this period and threshold be arbitrary. This allows the system administrator to determine what the appropriate values for each particular application are, since different systems have different requirements.

Anytime the system detects a possible attack, it makes a record of this attack and may block an attacker from accessing the system any further. Furthermore, an alert may be sent to the system administrator.

While the IDPS is scanning for attacks, it makes a log of access attempts into the system. This is critical, because if the system administrator wishes to determine if an attack is being attempted at a later time, he has the option of looking back at the access logs to see what input a suspected attack attempted to issue. The system administrator may subsequently block the user if he determines the user was launching an attack on the system. Furthermore, the system administrator or an analyst may examine the log to learn what strategies and patterns the attacker used. This invaluable information may be evaluated and used to develop new SQL-I attack signatures or tweak existing ones.

6. IDPS DESIGN AND IMPLEMENTATION

6.1 SQL-I Attack Detection

The IDPS system uses both signature-based and anomaly-based detection models to identify threats and attacks on the system. Signatures are carefully selected to implement the signature-based model. Anomaly-based detection is based on the number of times a user attempted to access the system, regardless of whether any SQL-I patterns have been detected.

IDPS is case-insensitive while trying to use the signature-based detection method to detect SQL-I attacks. The IDPS deals with White Space Manipulation attack by removing any white space before comparing text with known SQL-I attack patterns. The IDPS deals with Comments attack by looking for comment characters in the submitted text. The String Concatenation attack is dealt with by looking for the concatenation operation characters or CONCAT function. The keyword "UNION" is searched for by the IDPS in case an attack tries to perform the UNION Injection attack. The system will also look for binary, hexadecimal, and decimal characters in the submitted text to catch instances of this SQL-I attack variation. Sample patterns may be found in the SQLI_PATTERNS table described in the IDPS database schema.

Even when no SQL-I attack pattern is detected in the submitted form text, the IDPS monitors the frequency of the login attempts to implement the anomaly-based detection method. When the number of visits has exceeded a predetermined threshold, the system automatically blocks the visitor for a time.

It is significant to note that the screen the user sees when he or she has entered an incorrect password or when an SQL-I pattern is detected in the text matches. No

feedback is communicated to the user as to whether or not the system has detected an attack to ensure the system limits unnecessary information broadcast.

6.2 Storing and Blocking Attacks

Even when a text is deemed to not contain an SQL-I attack signature, the access attempt along with user IP address and username and password issued is logged into the VISITORS table. This table may be reviewed by the system administrator or an analyst at a later time for possible attacks or new attack patterns. Normal users usually have a more predictable number of login attempts, and this number may be determined by observing everyday activity of these users. In their attempt to determine the weaknesses of the system, attackers will attempt to access the system significantly more often. If a user attempts to access to the system more than an arbitrary number over a predetermined amount of time, which is stored in a value in the CONFIG table, this behavior is suspicious and hence the user is a potential threat. Thus, this user should be automatically blocked for a certain amount of time. The user may attempt to access the system after this period of time has elapsed. This block is recorded in the HACKERS table with the reason "LIMIT_EXCEED" to serve as a permanent record.

If an SQL-I attack is detected, the user's IP address, the text that matched the SQL-I pattern, the browser the user was using, and the time of the login attempt are all recorded in the HACKERS table as a permanent record. The attempt may be further reported by the IDPS by the use of an e-mail notification.

A user who is detected to have submitted text that matches SQL-I attack signatures are not immediately blocked. However, if the user submits text matching the SQL-I attack signatures more than a predetermined number of times, a value which is

stored in a value in the CONFIG table, then several things occur. The user is immediately added to the blacklist table BLOCKED_HACKERS. Access to the system is subsequently blocked. Finally, an alert in the form of an e-mail may be sent to the system administrator. If desired, the system administrator may choose to restore access to the blocked user at a later time.

6.3 Preprocessing form data before submission for processing

If a client login request passes all checks, before the IDPS passes the form data back to the normal login authentication system, it uses the PHP function mysql_real_escape_string() to be proactive and add an extra layer of protection. The function mysql_real_escape_string() escapes special characters such as EOF chars, quotes, backslashes, carriage returns, nulls, and line feeds in the value passed to it so that it is safer to place it in a mysql_query().[9]

6.4 E-mail Notifications

The system administrator has the option to set up an e-mail address to received e-mail alerts. The system administrator may adjust the exact e-mail address, the frequency of the e-mails, and the type of activity he wishes to be notified of in the administration panel.

6.5 Database Schema

The following is a listing of the database schema and tables used by the IDPS described. The names of the tables are generic to allow easy understanding, but in a real system, the table names may be obfuscated by renaming them to arbitrary names for further security.

6.5.1 Table: BLOCKED_HACKERS

Description: This is the IP "blacklist" table. Records the list of users to determine which users are blocked when attempting to access the system. The table includes values for IP masks, using % as the "wildcard" character.

Field	Type	Null	Default
ip	varchar(50)	No	None
timestamp	timestamp	No	CURRENT_TIMESTAMP

Table 6-1 BLOCKED_HACKERS table schema

6.5.2 Table: CONFIG

Description: Used to hold the arbitrary parameters for signature-based detection (type = "AUTOBLOCK_IP_TRIES") and anomaly-based detection (type = "LIMIT_HOUR" or "LIMIT_DAY").

Field	Type	Null	Default
type	varchar(20)	No	None
limitval	int(11)	No	None

Table 6-2 CONFIG table schema

Sample values for the CONFIG table are as follows:

type	limitval
LIMIT_HOUR	7
LIMIT_DAY	20
AUTOBLOCK_IP_TRIES	200

Table 6-3 Sample values for CONFIG table

6.5.3 Table: HACKERS

Description: Used to record the users detected to submit text that matches SQL-I attack signatures during signature-based detection.

Field	Type	Null	Default
id	int(11)	No	None
ip	varchar(50)	No	None
request	varchar(1000)	No	None
userBrowser	varchar(1000)	No	None
timestamp	timestamp	No	CURRENT_TIMESTAMP

Table 6-4 HACKERS table schema

6.5.4 Table: SQLI_PATTERNS

Description: Table which holds the SQL-I attack signatures.

Field	Type	Null	Default
create_date	timestamp	No	CURRENT_TIMESTAMP
pattern	text	No	None

Table 6-5 SQLI_PATTERNS table schema

Sample values for the SQLI_PATTERNS table are as follows:

create_date	Pattern
10/22/09 11:28 PM	'or'i'='i
11/12/09 1:51 AM	\/*
11/12/09 1:41 AM	;
11/2/09 11:51 PM	'or'[0-9a-z!@#$%^&*()\[\]{}]*'='[0-9a-z\[\]{}]*
4/1/10 3:35 PM	&
11/2/09 11:48 PM	||
11/2/09 11:24 PM	union
11/12/09 1:47 AM	--
11/12/09 1:51 AM	*\/
12/17/09 1:09 AM	'or'a'<>'b
12/17/09 1:09 AM	'or'[0-9a-z!@#$%^&*()\[\]{}]*'<>'[0-9a-z!@#$%^&*()\[\]{}]*
2/30/10 3:44 PM	"
2/30/10 3:46 PM	concat(
2/30/10 3:46 PM	char(
2/30/10 3:44 PM	+
2/30/10 3:50 PM	admin
5/01/10 1:23 AM	#
5/01/10 1:23 AM	drop
5/01/10 1:24 AM	create
…	…

Table 6-6 Sample values of SQLI_PATTERNS table

6.5.5 Table: VISITORS

Description: Records the users (visitors) that attempt to access the system. Users are recorded into this table regardless if there is an SQL-I attack pattern detected.

Field	Type	Null	Default
id	int(11)	No	None
ip	varchar(50)	No	None
username	varchar(100)	No	None
userBrowser	varchar(1000)	No	None
timestamp	Timestamp	No	CURRENT_TIMESTAMP

Table 6-7 VISITORS table schema

6.5.6 Table: IP_WHITELIST

Description: This is the IP "white list" table. Records the list of users to determine which users are always allowed access when attempting to access the system. The table includes values for white list IP masks, using % as the "wildcard" character.

Field	Type	Null	Default
ip	varchar(50)	No	None
timestamp	timestamp	No	CURRENT_TIMESTAMP

Table 6-8 WHITELIST table schema

7. IDPS COMPONENTS AND INTERFACE

The system designed and implemented in this project is a complete IDPS which uses a combination of the signature-based detection model and the anomaly-based detection model. To detect attacks, the IDPS contains a list of signatures. The system contains the following crucial components to provide the complete web server security against the SQL-I attacks.

7.1 Graphical User Interface

The system provides a browser-based interface for the server administrator to monitor the system for possible attacks. The administrator can review occurrences of attacks, block IP addresses, unblock IP addresses, and other actions on the GUI. The GUI contains the following interfaces.

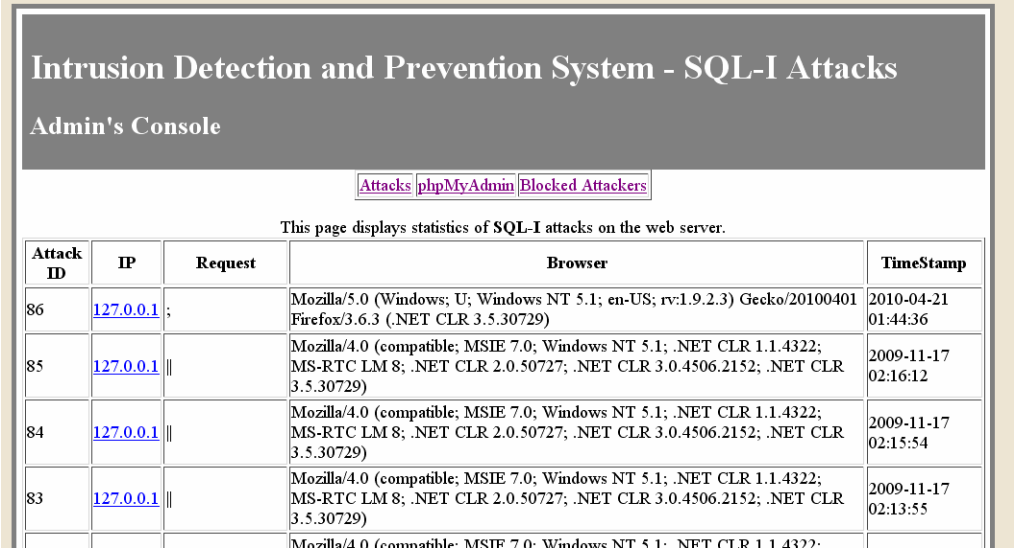
7.1.1 SQL-I Attacks (Administrator's) Console

The interface named "SQL-I attacks" lists all past attacks detected by the IDPS with the most recent ones listed first. This interface contains a table of attacks which contains the following attributes:

- Attack ID (iterative)
- Attacker's IP address linked to "Attacker History" interface
- The login name or password issued to perform the attack
- Browser that was used in the attack
- Timestamp of when the attack was detected

Clicking on an IP address entries takes you to the "Attacker History" interface page displaying the attack history of the clicked IP address. At the top of the page, there are

links to the “Blocked Attackers” page and “phpMyAdmin” interface. Figure 7-1 contains the screenshot of “CGI Attacks” interface.



Intrusion Detection and Prevention System - SQL-I Attacks

Admin's Console

Attacks | phpMyAdmin | Blocked Attackers

This page displays statistics of SQL-I attacks on the web server.

Attack ID	IP	Request	Browser	TimeStamp
86	127.0.0.1	;	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3 (.NET CLR 3.5.30729)	2010-04-21 01:44:36
85	127.0.0.1	||	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; MS-RTC LM 8; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)	2009-11-17 02:16:12
84	127.0.0.1	||	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; MS-RTC LM 8; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)	2009-11-17 02:15:54
83	127.0.0.1	||	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; MS-RTC LM 8; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)	2009-11-17 02:13:55

Figure 7-1 IDPS SQL-I Attacks (Administrator’s) Console Interface

7.1.2 Attacker History Interface

Clicking the IP address of the attack from the Attackers page brings up the “Attacker History” Interface page. The results on this page contain the attack history of the attacker. Information contained on this page include:

- Attacker’s IP address
- Total number of attacks that were detected
- “Block IP” button that blocks the IP address from accessing the server

Figure 7-2 illustrates this interface.

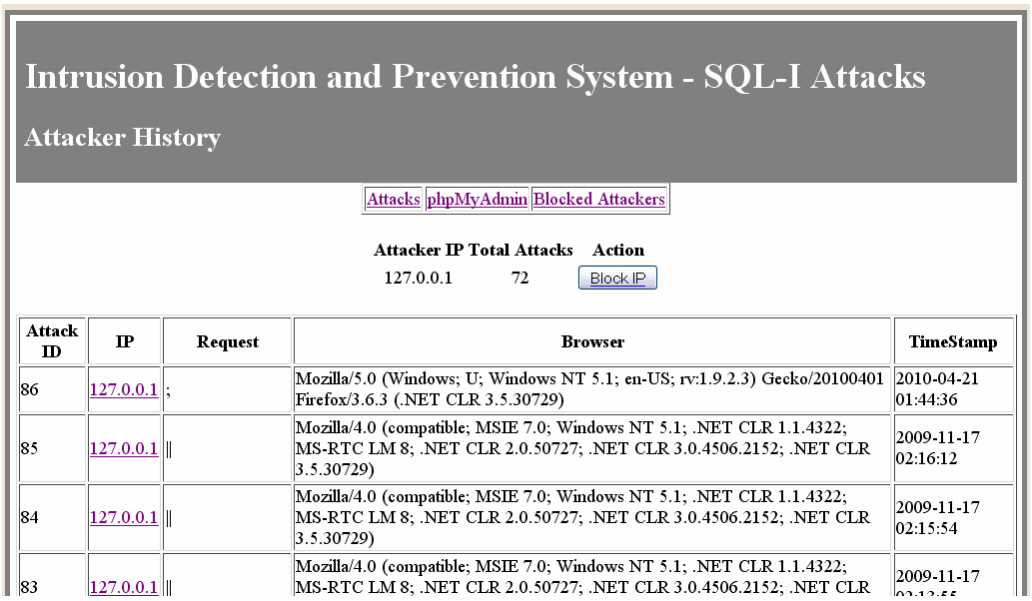


Figure 7-2 IDPS Attacker History Interface

Clicking on the Block IP button on the "Attacker History" interface page blocks the IP address from accessing the system and then immediately displays the "Blocked Attackers" interface page to the administrator.

7.1.3 Blocked Attackers Interface

If the system administrator blocked an IP from the Attacker History page, or if he clicked the "Blocked Attackers" link at the top of the page of most interfaces, then he will see the "Blocked Attackers" interface page.

The following are the attributes that are displayed on this page:

- the IP address blocked
- the timestamp of the block
- the option to unblock this IP address

Figure 7-3 illustrates this interface.

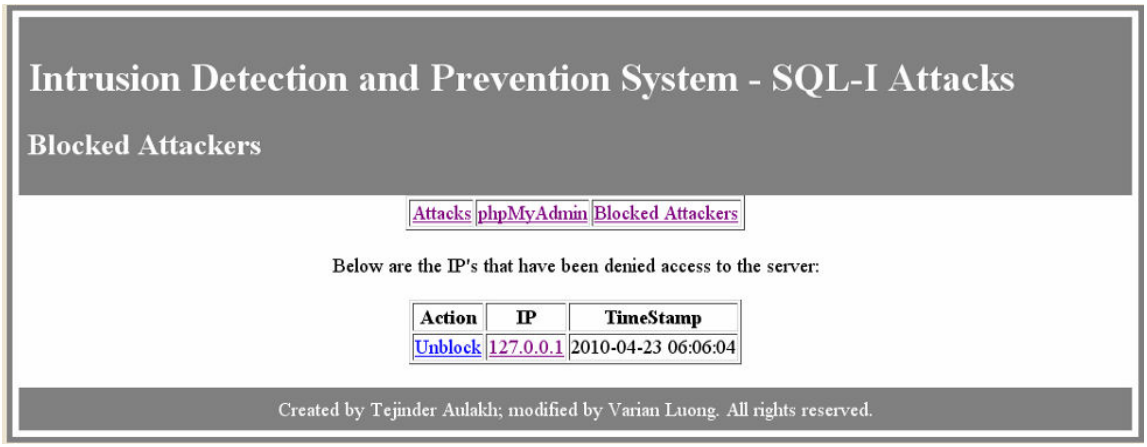


Figure 7-3 IDPS Blocked Attackers Interface

7.1.4 Block IP Address Mask Interface

In some cases where the system administrator suspects an attacker has a dynamic IP address, he may choose to block an entire subset of IP addresses. This is useful when the System Administrator detects an attack from a range of IP addresses and wishes to block the entire range.

Click on the Block IP Address Mask link at the top. In the resulting page, you will see the "Block IP Address Mask" interface. The following are the attributes that are displayed on this page:

- the IP address mask blocked
- the timestamp of the mask block
- the option to unblock this IP address mask

Next to IP mask pattern textbox, enter the pattern with % or * as the wildcard character. Finally, click the "Block IP mask" button to block the IP mask pattern. A message will then appear confirming the addition of this IP address mask to the list.

7.1.5 IP Address White List Management Interface

It is possible the system administrator wishes to designate IP addresses that he never wishes to be blocked by the IDPS system. These may be IP addresses from trusted locations, organizations, or individuals. Click on the White List Management link at the top. In the subsequent page, type in the IP address in the text box, then click on the "Add to White List" button to add the address to the white list. A message then appears to confirm that the IP address was added to the white list.

7.2 MySQL Database

MySQL is the DBMS that manages the tables that record login activity, attacker activity and history, SQL-I attack patterns, and IP address black and white lists.

7.3 Management Module

The management module is the primary component in the IDPS. It acts as the verification layer that checks the data that is passed between the web page form and the database.

7.4 E-mail Generation

Every time an SQL-I attack incident is detected, an e-mail is generated and sent to the administrator to ensure that the administrator is always kept informed about the attacks. The administrator can then choose to log into the system to perform further investigation into the suspicious activity and block the user. The e-mail contains the following information in the message:

- The IP address of the attacking machine
- The text used in the SQL-I attack
- The web browser used to perform the attack

Figure 7-4 below illustrates what the notification e-mail may look like.

Subject: "System alert: SQL-I pattern detected and logged"
From: SQL-I Detection System <varian@mail.com>
To: "varian@yahoo.com" <varian@yahoo.com>
Notice: A user with IP address 127.0.0.1 has
entered a username/password matching an SQL-I
pattern. This activity has been logged.

Details: Timestamp: = Sat Apr 10 8:44:36 UTC 2010
IP address = 127.0.0.1
username = ;
password = passw

Figure 7-4 Sample IDPS System Notification E-mail

8. IDPS REQUEST HANDLING FLOW CHART

When the form data is submitted to the web server, the data is forwarded to the SQL-I Intrusion Detection and Prevention System. The login attempt information — such as visitor IP address, time of login, and login username and password — is stored in the IDPS database for future reference. The IDPS checks to see if the visitor is on the Trusted List (also called the "white list"). If so, then the username and password is passed through the mysql_real_escape_string function to clean up the escape characters, and then processed normally. Otherwise, the IDPS checks to see if the visitor is marked Blocked (also called on the "blacklist"). If the visitor is, an error page is returned and the visitor is denied access to the system.

The IDPS will continue to check if the client attempting to log in has exceeded the threshold allowed per the time period on the system. If yes, then an error page is returned and the visitor is denied access temporarily.

Next, a scan of the submitted form text for SQL-I attack patterns is performed. If no attack pattern is found, then it is filtered using mysql_real_escape_string and processed normally. If an attack pattern was found, then the attacker's information is recorded into the database. If the occurrences of an attack are less than a threshold, then filter and process the data normally. However, if it is greater than the threshold, then the attacker information is stored in the database, the attacker is blocked indefinitely, an e-mail is sent to the system administrator, and an error page is displayed.

Figures 8-1a and 8-1b are flow charts that show how requests are handled by the IDPS, and how it intercepts and handles an SQL-I attack event.

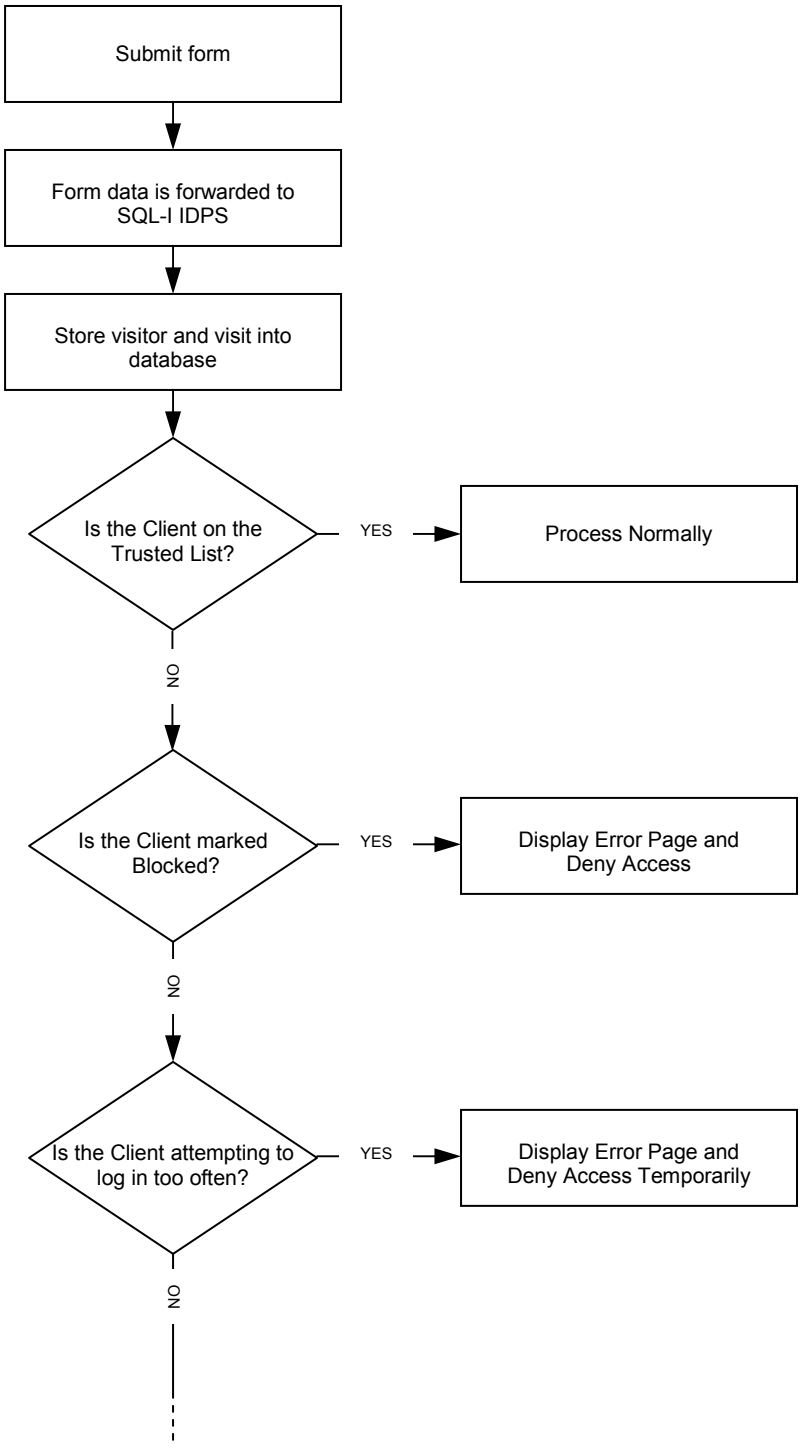


Figure 8-1a IDPS Request Handling Flow Chart

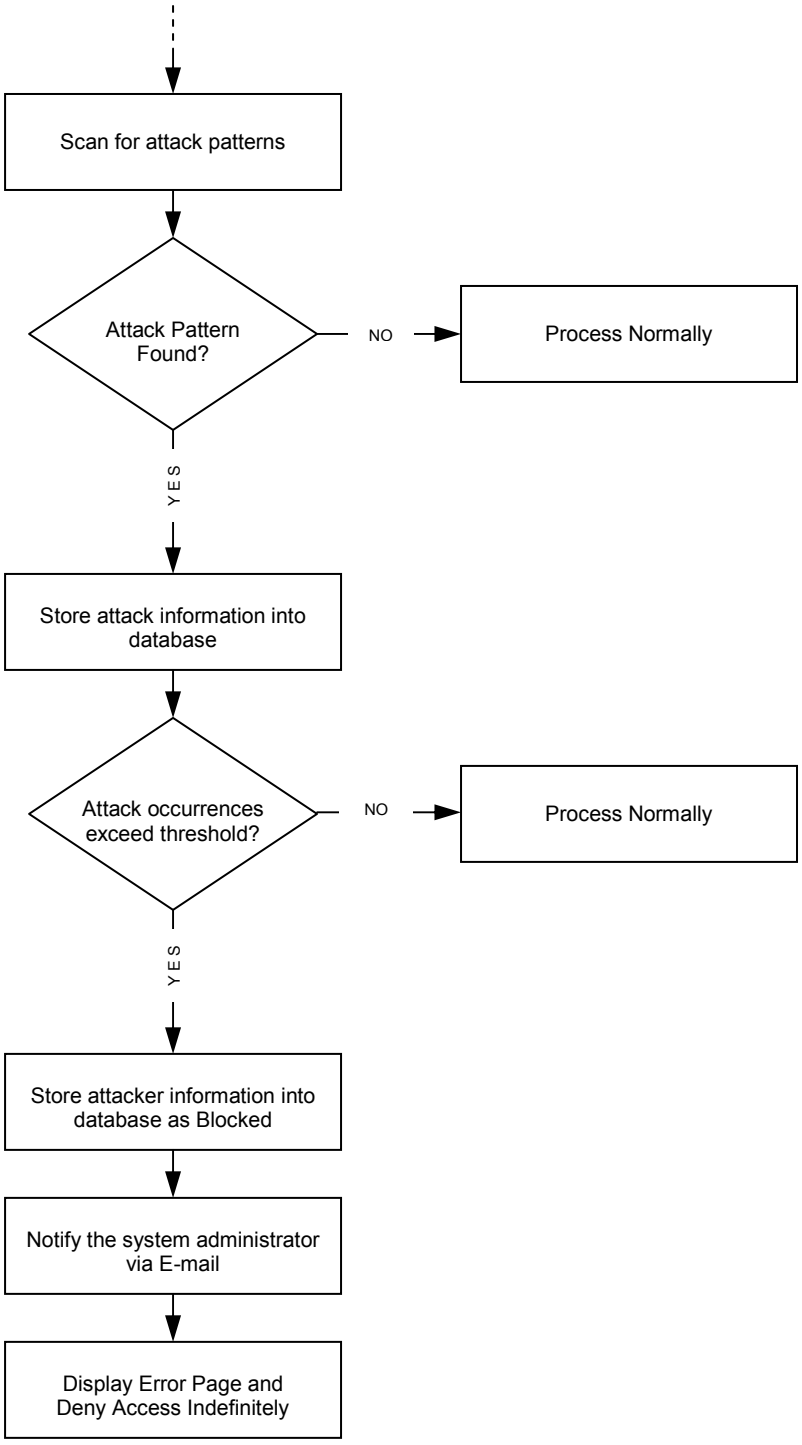


Figure 8-1b IDPS Request Handling Flow Chart (continued)

9 RESULTS AND OUTCOMES

The IDPS was tested by attempting to submit various known SQL-I attack patterns using a "test plan". The system successfully blocked the attacks and sent e-mail notifications to the server administrator. The system was tested on its ability to block and unblock attackers. Figure 9-1 and 9-2 illustrates one of these attempts to pass a username with a known SQL-I attack pattern and the resulting page that looks like a mundane invalid username/password page.

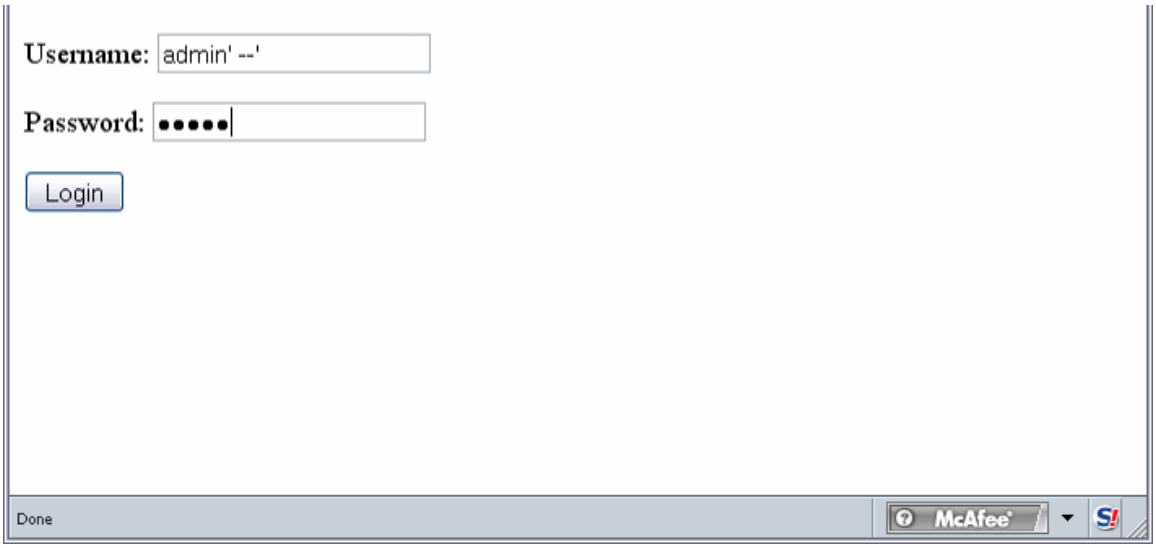


Figure 9-1 Login Page with SQL-I Attack Pattern

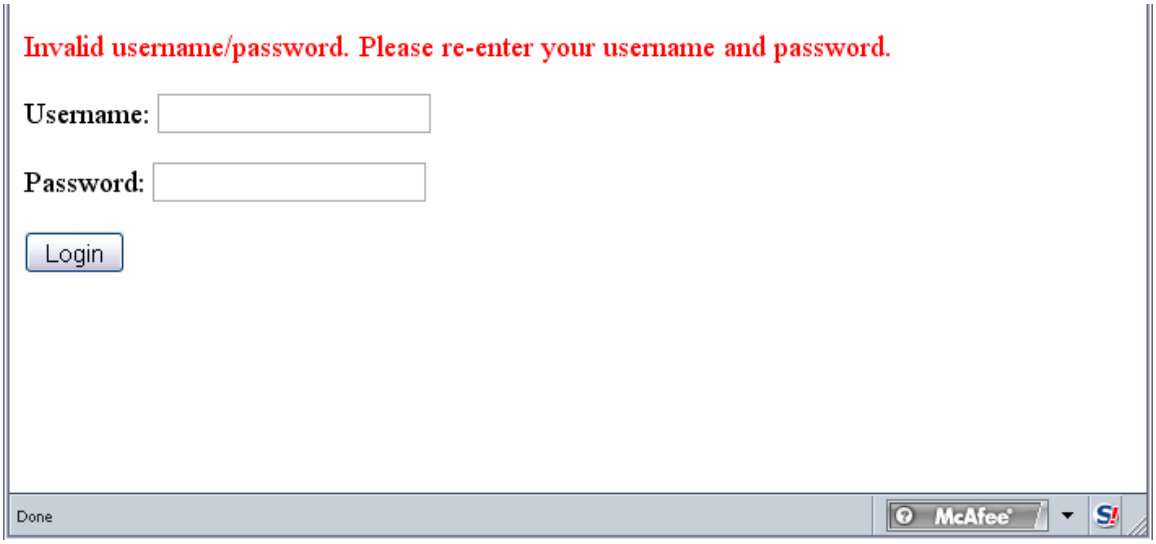


Figure 9-2 Resulting Error Page with Invalid Username/Password Message

A wide variety of input representing the different types of SQL-I attacks is determined and incorporated into a test plan that will be used to test the IDPS system. After running the complete test plan, we will determine the relative effectiveness and possible weaknesses of the IDPS in detecting SQL-I attacks on the system.

9.1 Test Plan and Results

The test plan consists of test cases, each of which contains a different input for the IDPS system, and represents a particular type of SQL-I attack pattern that may be used to attack the system. Each of these will be issued against the system. Then, the output and results will be recorded in the system to be analyzed and determine how effective the IDPS is. A brief excerpt of the test cases and results are outlined in table 9-3.

Attack Type	Username input	Password input	Result
AND/OR	admin	' OR '1'='1	Blocked
AND/OR	admin	' OR '0'<>'1	Blocked
White Space Manipulation Variation of AND/OR	admin	'OR'1'='1	Blocked
White Space Manipulation Variation of AND/OR	admin	'OR'0'<>'1	Blocked
Comments	admin'--	abcd	Blocked
Comments	admin'#	abcd	Blocked
Comments	admin'/*	abcd	Blocked
Comments	' or 1=1--	abcd	Blocked
Comments	' or 1=1#	abcd	Blocked
Comments	' or 1=1/*	abcd	Blocked
Comments	') or '1'='1--	abcd	Blocked
Comments	') or ('1'='1--	abcd	Blocked
Comments	/*!32302 10*/	abcd	Blocked
Comments	10; DROP TABLE members --	abcd	Blocked
Comments	10; DROP TABLE hackers2; /*	*/ OR 'abcd' <> '	Blocked
Comments	admin'); drop table hackers2; --	abcd	Blocked
Comments	admin'--	' OR '1'='1	Blocked
UNION	' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--	abcd	Blocked
UNION	' UNION select all from dummy --'	abcd	Blocked
String Concatenation	' UNI' + 'ON' + ' select "test" from dummy'	abcd	Blocked
String Concatenation	' UNI' || 'ON' || ' select "test" from dummy'	abcd	Blocked
String Concatenation	CONCAT(CHAR(65),CHAR(68),CHAR(77), CHAR(73),CHAR(78))	abcd	Blocked
Comments variation of UNION	' UNI/*breakUpThisKeyword*/ON select "test" from dummy'	abcd	Blocked
Hex/Bin/Dec	1 UNION SELECT ALL FROM WHERE	abcd	Blocked
Case-insensitivity, white-space manipulation	admin	U n I o N	Blocked

Table 9-3 Test Plan and Results Sample

9.2 Conclusion from Test Plan Results

A range of attack patterns were tested against the system according to the test plan outlined above. When sending an attack pattern to the server, the web server is able to successfully prevent the attack, log the attack entry in the database, and notify the administrator of the attack through e-mail. Further attacks are not possible because the attacker's IP address is subsequently blocked.

10 FUTURE WORK

The IDPS for SQL-I attacks was inspired by the future work listed by another thesis project that focused on CGI-script attacks.[6] Future consideration may be further extensions to the IDPS as suggested by this author.

- **PHP-related attacks**

  An extension to handle PHP attacks may be added to the system.

- **Protection for accessing private documents**

  An extension may be added to the IDPS to monitor and block attempts to access private documents on the system. The administrator of the system may choose which documents should be considered private.

- **SQL-I IDPS that automatically learns from previous attacks**

  It is conceivable that the SQL-I IDPS system can be programmed to take patterns from known attacks and automatically determine patterns that may be useful in preventing future attacks that use the same strategy. This feature will be a analytical engine that "learns" from past attacks.

- **Dealing with dynamic IPs**

  It is possible that an attacker is able to change IP addresses. Then our system will not be able to block all the IP addresses that the attacker is able to use. IDPS will be much more effective if it is able to deal with attackers that change their IP addresses.

11 CONCLUSION

This paper describes the challenges that Internet applications that make use of a database system face in terms of security and protection the private data. SQL-I attacks is a legitimate threat that endangers the confidentiality of data and may cost an organization a great deal of money and even their reputation.

Four different tools. The IDPS system developed for SQL-I attacks, when used properly, is an inexpensive and effective deterrent to hackers. It combines both signature-based detection and anomaly-based detection methods to effectively create a robust, secure system from SQL-I attacks. It is customizable and has the flexibility to be tuned to match the usage and needs of the system the IDPS protects. When combined with the original CGI-Script attack protection feature, the system is significantly safer from both online threats.

As long as PII is available on the Internet and cyber criminals can benefit from obtaining this information, there will be constant need to protect it.

REFERENCES

[1] Webpage “Wikipedia.org: SQL” http://en.wikipedia.org/wiki/SQL Retrieved on 2010-03-01.

[2] Webpage “SQL Injection Cheat Sheet” http://ferruh.mavituna.com/sql-injection-cheatsheet-oku Retrieved on 2010-03-01.

[3] Webpage “Homepage for GreenSQL” http://www.greensql.net/ Retrieved on 2009-12-15.

[4] Webpage “About page for dotDefender from Applicure” http://www.applicure.com/About_dotDefender Retrieved on 2009-12-15.

[5] Webpage “About page for CodeScan from CodeScan Limited” http://www.codescan.com/about-codescan/what Retrieved on 2010-04-10.

[6] Aulakh, T. Intrusion Detection and Prevention System: CGI Attacks, 2009. San Jose State University master’s thesis project.

[7] File: sql-injection-detection-wp.pdf Website: http://www.f5.com/pdf/white-papers/sql-injection-detection-wp.pdf Retrieved 2009-10-01

[8] Webpage “Imperva ADC Blind SQL Injection” http://www.imperva.com/resources/adc/blind_sql_server_injection.html Retrieved 2009-10-01

[9] Webpage “The PHP Manual: mysql_real_escape_string()” http://php.net/manual/en/function.mysql-real-escape-string.php Retrieved 2010-04-24.

[10] Kosuga, Y.; Kernel, K.; Hanaoka, M.; Hishiyama, M.; Takahama, Yu., "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection," Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual , vol., no., pp.107-117, 10-14 Dec. 2007

[11] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection," Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on , vol., no., pp.1411-1414, 27-29 April 2009

[12] Buehrer, G., Weide, B. W., and Sivilotti, P. A. 2005. “Using parse tree validation to prevent SQL injection attacks. In Proceedings of the 5th international Workshop on Software Engineering and Middleware” (Lisbon, Portugal, September 05 - 06, 2005). SEM '05. ACM, New York, NY, 106-113.

[13] Webpage “Blue Shield of California – Provider Connection: Electronic Data Interchange” https://www.blueshieldca.com/provider/common/eclaims_quart_1_8.jhtml Retrieved 2010-05-03.