**BUSINESS IMPACT ANALYSIS**

The SQL injection attack consists of the insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

**Threat Modelling**

* SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
* SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.
* The severity of SQL Injection attacks is limited by the attacker’s skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

**Conduct a thorough analysis of the potential business impact of SQL injection vulnerability**

SQL injection attack occurs when untrusted data enters a program and is used to dynamically construct a SQL query. The consequences of such attacks can affect confidentiality, authentication, authorization, and integrity.

* Confidentiality: SQL injection vulnerabilities can lead to the loss of confidentiality as attackers can access sensitive data stored in SQL databases. This can result in the exposure of valuable or confidential information, leading to various risks such as fraud or identity theft.
* Authentication: Poorly constructed SQL commands used for checking user credentials can be exploited to gain unauthorized access to a system. Attackers can potentially connect to a system as another user without knowing their password, compromising the authentication process.
* Authorization: SQL injection vulnerabilities can allow attackers to manipulate authorization information stored in a SQL database. By exploiting these vulnerabilities, attackers can modify or change authorization settings, potentially granting themselves unauthorized privileges.
* Integrity: In addition to accessing sensitive information, SQL injection attacks can also modify or delete data in databases. Attackers can manipulate SQL queries to make unauthorized changes to the data, compromising the integrity of the system.

**Risk Factors:**

The platform affected can be:

Language: SQL

Platform: Any (requires interaction with a SQL database)

SQL injection attacks commonly affect database-driven websites and software packages. These vulnerabilities are easily detectable and exploitable, making any site or software with even a minimal user base susceptible to such attacks.

**Examples**:

**Example 1** demonstrates a SQL injection attack where an attacker provides malicious input, resulting in an SQL query with incorrect syntax. This can lead to unintended consequences and potential security breaches.

In SQL: select id, firstname, lastname from authors

If one provided: Firstname: evil'ex and Lastname: Newman

the query string becomes:

select id, firstname, lastname from authors where firstname = 'evil'ex' and lastname ='newman'

which the database attempts to run as:

Incorrect syntax near il' as the database tried to execute evil.

A safe version of the above SQL statement could be coded in Java as:

String firstname = req.getParameter("firstname");

String lastname = req.getParameter("lastname");

// FIXME: do your own validation to detect attacks

String query = "SELECT id, firstname, lastname FROM authors WHERE firstname = ? and lastname = ?";

PreparedStatement pstmt = connection.prepareStatement( query );

pstmt.setString( 1, firstname );

pstmt.setString( 2, lastname );

try

{

ResultSet results = pstmt.execute( );

}

**Example 2** showcases a dynamic SQL query constructed in C# code. Due to the concatenation of user input, the query becomes vulnerable to SQL injection. An attacker can manipulate the query to bypass security checks and access unauthorized data.

...

string userName = ctx.getAuthenticatedUserName();

string query = "SELECT \* FROM items WHERE owner = '"

+ userName + "' AND itemname = '"

+ ItemName.Text + "'";

sda = new SqlDataAdapter(query, conn);

DataTable dt = new DataTable();

sda.Fill(dt);

...

The query that this code intends to execute follows:

SELECT \* FROM items

WHERE owner =

AND itemname = ;

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string "name' OR 'a'='a" for itemName, then the query becomes the following:

SELECT \* FROM items

WHERE owner = 'wiley'

AND itemname = 'name' OR 'a'='a';

The addition of the OR 'a'='a' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

SELECT \* FROM items;

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

**Example 3** highlights the effects of a malicious value passed as input. The attack string can execute multiple SQL statements, allowing the attacker to delete data from the database or execute arbitrary commands. The use of comment characters can further manipulate the query to achieve the desired outcome.

If an attacker with the user name hacker enters the string "name'); DELETE FROM items; --" for itemName, then the query becomes the following two queries:

SELECT \* FROM items

WHERE owner = 'hacker'

AND itemname = 'name';

DELETE FROM items;

--'

Many database servers, including Microsoft® SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error in Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, in databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. In a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in Example 1. If an attacker enters the string "name'); DELETE FROM items; SELECT \\* FROM items WHERE 'a'='a", the following three valid statements will be created:

SELECT \* FROM items

WHERE owner = 'hacker'

AND itemname = 'name';

DELETE FROM items;

SELECT \* FROM items WHERE 'a'='a';

**Understanding the potential consequences of SQL injection vulnerability on a business**

SQL injection vulnerabilities can have significant business impacts on organizations, including financial losses, damage to reputation, legal consequences, and operational disruptions. Let's conduct a thorough analysis of these potential impacts:

**Financial Losses**:

Unauthorized access to sensitive data: SQL injection can allow attackers to retrieve, modify, or delete data stored in databases. If attackers gain access to valuable or confidential information such as customer data, financial records, or intellectual property, it can lead to financial losses through fraud, identity theft, or competitive disadvantages.

Loss of revenue and business opportunities: Exploitation of SQL injection vulnerabilities can result in service disruptions, leading to lost sales, customers, and business opportunities.

Cost of incident response: Organizations may need to invest in forensic investigations, security assessments, and remediation efforts to mitigate the impact of a SQL injection attack. These costs can be significant, depending on the severity and scope of the incident.

Damage to Reputation: Customer trust and brand reputation: If a SQL injection vulnerability is exploited, customer data may be compromised, eroding trust in the organization's ability to protect sensitive information. News of a data breach can spread quickly, damaging the organization's reputation and leading to customer churn.

Compliance and regulatory issues: Many industries have legal requirements and regulations regarding the protection of customer data. A SQL injection attack resulting in data breaches can lead to non-compliance penalties, fines, and legal actions.

**Legal Consequences**:

Legal liabilities: Organizations may face legal consequences, including lawsuits from affected customers or stakeholders, for failing to protect their data adequately. This can result in significant financial penalties and legal fees.

Regulatory fines: Government agencies, such as data protection authorities, may impose fines for data breaches caused by SQL injection vulnerabilities, particularly if non-compliance with applicable regulations is established.

**Operational Disruptions:**

Downtime and service disruptions**:** Successful exploitation of SQL injection vulnerabilities can lead to system downtime or reduced availability of critical applications or services. This can impact business operations, productivity, and customer satisfaction.

Recovery and remediation costs: Resolving a SQL injection attack requires investigation, vulnerability patching, data recovery, and infrastructure reconfiguration. These activities can be time-consuming, disrupt normal operations, and incur additional costs.

**Assessing the risk to the business**

Assessing the risk of a SQL injection involves evaluating various factors to determine the likelihood and potential impact of an attack. Here are key aspects to consider:

Application Exposure:

* Input points: Identify all the input sources where untrusted data enters the application, such as user inputs, form fields, query parameters, or API endpoints.
* Query execution: Understand how the application constructs and executes SQL queries using the input data.

Vulnerability Severity:

* Attack surface: Determine the extent of SQL injection vulnerabilities present in the application. Assess if there are any known vulnerabilities in the underlying frameworks, libraries, or database management systems.
* Vulnerability complexity: Consider the complexity required for an attacker to exploit the SQL injection vulnerability. This includes factors like input sanitization techniques, query construction methods, and server-side protections.

Data Sensitivity:

* Identify the sensitivity of the data stored and accessed by the application. Evaluate the potential impact if an attacker gains unauthorized access, modifies, or deletes sensitive data.
* Consider the compliance requirements for protecting certain types of data, such as personally identifiable information (PII), financial records, or intellectual property.

Threat Landscape:

* Assess the likelihood of an attacker attempting a SQL injection attack based on industry trends, the application's visibility, and historical attack patterns.
* Consider the presence of automated tools or scanners that can identify SQL injection vulnerabilities and the likelihood of attackers utilizing such tools.

Security Controls:

* Evaluate the effectiveness of existing security controls, such as input validation mechanisms, prepared statements or parameterized queries, and secure coding practices.
* Assess the application's security monitoring capabilities, including the ability to detect and respond to SQL injection attempts or unusual database activities.

Business Impact:

* Consider the potential financial, reputational, legal, and operational consequences discussed in the previous analysis.
* Evaluate the value of the application, the criticality of its functions, and the potential impact of an extended downtime or compromised data.

Based on the assessment, assign a risk rating or level to the SQL injection vulnerability, such as low, medium, or high. This rating can help prioritize mitigation efforts and allocate appropriate resources for addressing the vulnerability.

It is important to note that risk assessments should be conducted regularly as the threat landscape evolves, new vulnerabilities emerge, and the application undergoes changes or updates. Ongoing monitoring, vulnerability scanning, and penetration testing can help identify and mitigate SQL injection risks effectively.