# HACKER PREVENTION GROUP ASSIGNMENT

| **Security Concern #** | **SQL Injection** |
| --- | --- |
| Contributing Team Members: | Lyea Paul |
|  | Jinil Parekh |
|  | Navaneeth Krishnan |
|  | Pavan Mistry |

## DEFINITION:

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database allowing an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks.

OBJECTIVE OF ATTACK:  
A successful SQL injection attack can result in unauthorized access to sensitive data, such as:

* Passwords.
* Credit card details.
* Personal user information.

SQL injection attacks have been used in many high-profile data breaches over the years. These have caused reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period

There are lots of SQL injection vulnerabilities, attacks, and techniques, that occur in different situations which include

* Retrieving hidden data, where you can modify a SQL query to return additional results.
* Subverting application logic, where you can change a query to interfere with the application's logic.
* UNION attacks, where you can retrieve data from different database tables.
* Blind SQL injection, where the results of a query you control are not returned in the application's responses.

(<https://portswigger.net/web-security/sql-injection#sql-injection-examples>)

NOTABLE CASES/VICTIMS:  
The GhostShell gang on Saturday posted online what it claims are accounts and records from various financial services, consulting firms, academia, law enforcement, and the CIA. "Team GhostShell final form of protest this summer against the banks, politicians and for all the fallen hackers this year," the post said in part. "One million accounts/records leaked. We are also letting everyone know that more releases, collaborations with Anonymous and others, plus two more projects are still scheduled for this fall and winter. It's only the beginning."

Researchers at Imperva say the attackers appear to have employed mostly SQL injection, but also exploited weak passwords and vulnerable content management systems. The attackers used the popular SQLmap tool, and some of the hacked databases included more than 30,000 records.  
(<https://www.darkreading.com/cyberattacks-data-breaches/ghostshell-haunts-websites-with-sql-injection>)

## EXAMPLES:

There are many ways to hack a database using SQL Injection.

1. **Line comments**

Line comment sql injection works by putting a line comment at the end to comment out the rest of the query. Line comments are typically used to ignore the rest of the original query so you don’t need to worry about valid syntax.

Example: logging in as admin:Injection into the username parameter: admin'--

SELECT \* FROM members WHERE username = '**admin'--**' AND password = 'password'   
If successful, this will log you as the admin user because the rest of the SQL query after -- will be ignored.

1. **Inline comments**

You can use inline comments to comment out the rest of a query as with line comments (by simply not closing the comment). They are also useful for manipulating characters to bypass filtering/blacklisting, remove spaces, and obfuscate queries. In MySQL, you can use its special comment syntax to detect the database and version.

Example: Comment SQL Injection Attack Samples

ID value: 10; DROP TABLE members /\*

Simply get rid of other stuff at the end of the query. Same as 10; DROP TABLE member--

SELECT /\*!32302 1/0, \*/ 1 FROM tablename

Will throw an division by 0 error if MySQL version is higher than 3.23.02

1. **Stacking Queries**

Stacking means executing more than one query in one transaction. This technique can be very useful but only works for some combinations of database server and access method:

Example: SELECT \* FROM members; DROP members--

When successful, this will end one query and start another one.

Stacked SQL Injection Attack Samples

ID value: 10;DROP members --

SELECT \* FROM products WHERE id = 10; DROP members--

This will run DROP members SQL sentences after normal SQL Query.

1. **If Statements**

Get a response based on an IF statement. This is one of the key techniques for Blind SQL Injection. Also very useful to test simpler things blindly yet accurately.

Example: if ((select user) = 'sa' OR (select user) = 'dbo') select 1 else select 1/0 (S)

This will throw a divide by zero error if the user currently logged in is not sa or dbo.

1. **Bypassing login screens**

This is SQL injection 101—here are some typical login tricks that you can use with form fields and parameters:

Example: SELECT \* FROM users WHERE username = '**'' or ''='**' AND password = '**'' or ''='**'

admin' --

admin' #

admin'/\*

' or 1=1--

' or 1=1#

' or 1=1/\*

') or '1'='1--

') or ('1'='1--

Another trick is to log in as a different user:

' UNION SELECT 1, 'anotheruser', 'doesn't matter', 1–

([source:https://www.invicti.com/blog/web-security/sql-injection-cheat-sheet/#LineCommentAttacks](https://www.invicti.com/blog/web-security/sql-injection-cheat-sheet/#LineCommentAttacks))

## PREVENTION:

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes.

Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.

**Step 1: Train and maintain awareness**

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

**Step 2: Don’t trust any user input**

Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.

**Step 3: Use whitelists, not blacklists**

Don’t filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only.

**Step 4: Adopt the latest technologies**

Older web development technologies don’t have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, PHP uses PDO instead of MySQLi.

**Step 5: Employ verified mechanisms**

Don’t try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.

([source: https://www.acunetix.com/websitesecurity/sql-injection/](https://www.acunetix.com/websitesecurity/sql-injection/))