

VISA Europe SQL Injection Attacks

Data has become the preferred currency of today's card criminals. Their aim is to obtain sensitive personal and account data. And, with so many online retailers routinely collecting and storing this type of information, the e-commerce space has become a particular target.

Visa Europe is therefore eager to engage with the e-commerce community. By raising awareness of the vulnerabilities and offering practical advice and guidance, we can help to protect all parties from the risks and costs of fraud.

Understanding and avoiding SQL injection attacks

One of the most prevalent ways for criminals to obtain account data is via a so called "SQL Injection" attack.

SQL is the name of the programming language used by database systems. Whenever a customer provides their personal data to an online retailer (by filling in an online form, for example), an SQL query is created to talk to the retailer's database. With a basic knowledge of SQL, criminals have become adept at remotely infiltrating vulnerable systems and extracting customer data.

When working towards compliance with the Payment Card Industry Data Security Standards (PCI DSS), or maintaining compliance, protection against SQL Injection attacks should therefore be regarded as a prime consideration.

To support increased payment security, Visa Europe is providing best practices to assist merchants and other stakeholders in protecting against common causes of system breaches. Whilst every reasonable effort has been made to ensure the accuracy of information provided by Visa Europe, Visa Europe shall not be held liable for any inaccurate information of any nature, however communicated by Visa Europe.

Mitigating Techniques

There are a number of relatively simple steps that can be taken to significantly reduce an organisations exposure to SQL injection attacks. Please note, these measures will not completely eliminate the risk involved in processing user input but will help make the system considerably more secure. As SQL injection attacks are a result of placing too much trust in user input, the guiding principle in mitigating SQL injection attacks is to ensure that all input sources are validated.

The table overleaf describes common techniques to limit SQL injection vulnerabilities and should be followed when implementing any web presence which requires the support of a database system. Please note that many of these controls should be implemented in conjunction with each other (rather than in isolation) to form a layered defence to SQL injection.



Constrain User Input

Input Validation

Relatively simple checks of user input can prevent many SQL attacks. For example, in the case of an expected numeric input, a developer can simply reject any input that contains characters other than digits. Developers should establish input validation routines that identify good input and reject everything else. Writing regular expressions to filter input can be complicated; however, there are numerous freely available resources on the internet to assist developers in writing regular expressions (see Appendix A).

Escaping User Supplied Input

Each database management system supports a character escaping method where a developer can indicate that the data being passed to the database is intended to be evaluated as data and not executed as code. Effective use of escaping can drastically limit the exposure of a web server to SQL injection attacks.

Use Stored Procedures and Parameterised Queries

Stored Procedures

A stored procedure is a set of SQL statements with an assigned name that resides in the database in a compiled form. Stored procedures provide a first line of defence in protecting against SQL injection; however, it is a common misconception that using stored procedures by themselves can render a system invulnerable to SQL injection attacks.

SQL injection may still be possible, especially if dynamic SQL is not handled appropriately within a stored procedure. As a general rule, generating dynamic SQL inside a stored procedure should not be allowed. Where unavoidable, additional techniques such as escaping user input, input validation checks and parameterised queries must be used.

Parameterised Queries

Parameterised queries allow a developer to set up a SQL statement once, and then execute that statement many times using different parameters. The use of prepared statements should be the corner stone of developing SQL injection resilient code and be used in conjunction with stored procedures. Where possible, all queries should be parametrised and all dynamic data should be explicitly bound to parametrised queries. In addition, if the application relies on the use of dynamic queries, the application must avoid the use of string concatenation in their construction.

Note: both stored procedures and prepared statements may be vulnerable to SQL injection attacks unless developers rigorously apply defensive coding practices to all possible user-supplied input.

Additional Controls

Principle of Least Privilege

A successful SQL injection attack enables a malicious user to execute commands using the database privileges granted to the application. If the application uses an over-privileged account to connect to the database, an attacker can abuse these privileges. A web server should never run as root or access a database as administrator, a database in turn should have minimal privileges on the underlying Operating System.

System Hardening

In general, complexity is the enemy of good security and, as such, only necessary services and protocols should be supported, all other functionality should be disabled. In addition, all default accounts should be either changed or removed as a matter of priority.

Technical Guidance on how to harden IIS and Apache web servers, as well as underlying Operating Systems, can be found at the Centre for Internet Security (see Appendix A).

Error Messages

The application must avoid disclosing detailed error information to the end user. Malicious users often use the information contained in error messages to discover information about the system such as table and database naming conventions. As this information can be used to refine an attack against the system, the application should never pass back detailed diagnostic information to the end user.

Code review

Secure code reviews should form an integral part of any application development process. These reviews should focus on identifying insecure coding techniques and vulnerabilities that could lead to the realisation of a security threat (i.e. information leakage, integrity violation, denial of service and/or illegitimate use). Identifying and mitigating vulnerabilities early in the development lifecycle can be significantly more cost effective than retroactively finding and fixing vulnerabilities later in the application cycle. Remember, all it takes is one lazy programming practice or shortcut in how input is validated to (re)introduce a SQL vulnerability into a system.

Web Application Firewalls

Web Application Firewalls provide an additional layer of defence in protecting systems against various methods of attack. While application firewalls can reduce the likelihood of a compromise, they are not a panacea for protecting systems against SQL injection. The most effective long term mitigation technique will always be ensuring application code correctly handles user input.

Appendix A - Additional Resources for Developers

The Open Web Application Security Project

- Testing for SQL Injection - [http://www.owasp.org/index.php/Testing_for_SQL_Injection_\(OWASP-DV-005\)](http://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005))
- SQL Injection Prevention Cheat Sheet - http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- Guidance for Performing Code Reviews - http://www.owasp.org/index.php/OWASP_Code_Review_Guide_Table_of_Contents

Centre for Internet Security

- Provides guidance on system and application hardening - <http://www.cisecurity.org/benchmarks.html>

Regular Expressions

- Guidance on writing regular expressions and how to avoid common pitfalls - <http://www.regular-expressions.info/>.