



NISCC Technical Note 09bis/04

Issued 1 November 2004

Guidance on Mitigating the Security Risks of SQL Injection Attack

Key Points

- SQL injection attacks are a serious risk to SQL database front-ends, and potentially the systems behind them.
- Good security practices to protect SQL applications from “bad” user data are required.
- Measures to limit the information returned to a malicious user should be considered.
- With good security by system designers, the risk from SQL injection attack can be minimised.

National Infrastructure
Security Co-Ordination Centre
PO Box 832
London
SW1P 1BG

Tel: 020 7821 1330 Ext 4511
Fax: 020 7821 1686
Email: enquiries@nisc.gov.uk
Web: www.nisc.gov.uk

This document was written by a working group of the UK Network Security Information Exchange.

AIM

1. This NISCC Technical Note is intended to provide guidance to the target audience of database owners on the mitigation of security risks from SQL injection attack.

TARGET AUDIENCE

2. This document addresses several target audiences, the principle controls applicable to each audience are parenthesised.

- Application Security Architects (design standards)
- Coding Staff (implementation standards)
- Supply Chain (contractual terms)
- Operational Security (platform and deployment standards, monitoring practices)

INTRODUCTION

3. SQL injection is a technique that can be used to gain read, write and execute rights initially to the database and by extension to the underlying operating system and network.

4. All applications that connect to a database and accept dynamic user input in the context of the database are vulnerable to attack. If this user input is entered in a carefully crafted format the attacker may be able to manipulate the SQL commands passed to the database so that the commands executed are different from what the developer originally intended.

5. External web based applications provide the most commonly understood channel of attack. Web applications that provide an interface to 'On-Line Transaction Processing' (OLTP) systems are very likely to be targeted.

IMPACTED TECHNOLOGIES

6. Almost all SQL based database systems (Oracle, DB2, SQL Server etc) are vulnerable to SQL injection attacks. This is due to the semantics of SQL, rather than the database implementations. The type of attacks to which a database is vulnerable depends on the functionality provided by the database and the way the provider has implemented SQL.

DEFENCE IN DEPTH

7. The mitigation of SQL injection should be set in the context of a defence in depth strategy. This means applying appropriate security controls firstly to stop an attack from taking place and secondly to minimise impact if an attack is possible. In the latter case, the principle of least privilege should be applied so

that applications, databases, operating systems and the network may only be accessed in line with businesses requirements. Other vulnerabilities in these areas should also be mitigated.

8. A typical SQL injection attack has three stages: gaining unauthorised access; modification of a database; and compromise of host. An illustrative Case Study of an attack, is available from your normal NISCC contact.

RISK ASSESSMENT

9. It is important for organisations to understand the nature of their exposure to SQL injection attacks. It is important to understand the basic elements of risk: impact, vulnerability and threat. Risk can be defined as the combination of the probability of an event occurring and the consequences of its occurrence.

10. Each of these can be mitigated:

- Impact, by early attack detection and appropriate backup measures;
- Vulnerability, by additional security measures;
- Threat is the hardest to mitigate, but profile as a target can be minimised in various ways (Don't be a "soft" target; good personnel security practices to reduce the insider threat etc).

11. A useful exercise is to assume that an SQL injection attack had been successful. What data has been compromised? What security measures are preventing further access to the host network? How much further could access be extended? What use can be made of the data or access gained? Who might have an interest in that result? How would such an attacker be likely to have mounted the attack - externally or internally? How quickly would the attack be noticed? What would be done to limit the damage? What would be the remaining cost?

12. When considering the risk posed by successful SQL Injection attacks, it is important to also consider the patch level of the affected database server. A number of database systems have been found to be vulnerable to buffer overflow attacks that can be triggered when a user can submit a query of their choice. Considering these 'secondary' issues may alter the risk assessment.

13. If a system is attached to a public network (Internet) then the possibility of an attack is high and given that the system is vulnerable the consequences in terms of data integrity, confidentiality and availability is high. Internal and Extranet applications should be considered as being scarcely less important, especially if there is scope for social engineering attacks (e.g. the misuse of remote access).

MITIGATION

14. To mitigate SQL injection attacks a number of measures singly or in conjunction may be employed. These measures fall into two categories;

measures that stop SQL injection attacks from taking place; and measures that minimise impact if arbitrary SQL can be injected.

15. During design and implementation, consider each input field individually, rather than having one policy for all. The designer/coder should firstly decide what is valid input for each input field. They should then consider how this might open up the application to SQL injection attack and add code to mitigate this. In an ideal development environment, the coders will have generic code/classes that they can re-use.

16. When analysing an application, it is important to consider all the possible inputs to the application. Examples include: form fields (including hidden fields); parameters in the query string; field/pair values in cookies; HTTP header parameters; and even data directly from the database (second order SQL injection).

17. Limiting dynamic SQL in the production environment will reduce the risk of an SQL injection attack. If dynamic SQL can be eliminated altogether, then the application will be immune to SQL injection attacks. However, this will not be possible in most applications and input validation will need to be added.

18. Validate all input so that the user-supplied data conforms to the expected data type of the field. For example, if an integer is anticipated only accept numbers as input.

19. Validate all input so that the user-supplied data does not contain the characters used as string delimiters in the SQL commands. The characters most commonly used as string delimiters are single quotes, though some implementations may use (for example) double quotes. Regular expressions should be used to detect these characters and reject input that contains these characters. Even with this validation, an attacker might inject the string delimiter character in another way, such as with the SQL CHAR function. Validation should also be added to stop these input attacks. Validate input for other dangerous characters and strings, such as semicolons and double dashes, in the same way.

20. For some input fields, such as a comment field, it will not be practical to reject dangerous characters altogether. In these cases, the dangerous character could be replaced with a substitute character or string. An alternative to this is to encode the user data (e.g. as hex or base-64). Parameterised queries in APIs such as Microsoft ADO, will encode the user data for you, however, safe APIs might not be available for some legacy languages. The string delimiter can also be escaped, by (for example) doubling up the single quote. However, this might not stop SQL injection altogether, as an attacker might use length limits by, for example, removing one of the doubled up single quotes.

21. Using stored procedures in the database will stop most SQL injection attacks. This is because the input parameters are treated as data and therefore cannot be used to alter the SQL statements within the stored procedure. However, if stored procedures are called dynamically (e.g. `strSQL = sp_InsertName`

'John'), SQL injection may still be possible, as an attacker can still exert influence over the command. For example, entering `John';drop table x--` will call the stored procedure and then drop the table 'x'. Stored procedures that call stored procedures dynamically using user-supplied data are also vulnerable to SQL injection.

22. Using stored procedures, encoding user-data and escaping user-data can lead to second order SQL injection. Although the original user-data was passed to the database in a safe manner, the data is actually stored in its original form (without the encoding or escaping characters). If this data is used in secondary SQL commands without validation, then SQL injection is possible. Data from the database should not therefore be automatically trusted.

23. Limiting the length of acceptable input will defeat many attacks if SQL injection is possible. For example, usernames entered by the user should not be longer than the maximum of characters that a username can contain. The scope for attack will therefore be restricted, as the attacker may not have enough characters to add further SQL commands. However, simple attacks such as bypassing form identification would still be possible.

24. Consider carefully the content returned by error messages to the user in their raw format with a view to modifying or suppressing the messages. This will restrict information-gathering attacks, which allow the attacker to gain knowledge of the names of tables, attributes and other database components. This information is extremely useful in facilitating an attack. However, modifying or suppressing error messages alone will not stop a determined attacker, as more advanced information gathering attacks exist, such as time-delay attacks.

25. Applying the principle of least privilege in the database will minimise impact if an attack is possible. End users should not be allowed to execute queries indiscriminately and they should only have read/write access to the resource that they need. And of particular concern are the SQL extended stored procedures, which allow the execution of system commands. Using linked databases (with pre-authenticated links) should also be carefully considered, as compromising one database could result in the compromise of others.

APPLICATION SECURITY ARCHITECTS

26. In the design phase of a project, application security architects should ensure that the security standards and architecture that they choose make the application invulnerable to SQL injection. The standards should incorporate the applicable mitigation measures from above. For the architecture, a three-layer model could be employed, where by the middleware components firstly validate the user input and secondly ensure integrity in the event of a compromise.

CODING STAFF

27. Coding staff must ensure that they adhere to the security standards for mitigating SQL injection attacks, which were selected during the design phase.

SUPPLY CHAIN

28. If application code is provided by a third-party supplier, the supplier must guarantee that they will provide a fix for security bugs in a timely fashion. Although no supplier is likely to provide a warranty that their code is bug-free, it is important to ensure that any security bugs will be fixed and that the fixes will be provided within some specified timescale.

29. When outsourcing development to a third-party supplier, request that they follow your security standards in development. This can be added to the agreement between yourselves and the third-party supplier.

OPERATIONAL SECURITY

30. Operational security must ensure that their applications are not vulnerable to SQL injection attacks. This can be achieved by thoroughly testing all new applications and all revisions to existing applications. The auditing process can be significantly reduced by using applications scanners, and there are both commercial, and open-source tools available. However, for mission critical systems, a more detailed manual inspection would be recommended in addition to the automated scanning.

31. Operational security must also ensure that least privilege is applied, as described in the mitigation section. In addition to this, IDS systems can be used to try and detect SQL injection signatures, as and when the attacks occur. However, the amount of false positives generated by these signatures might be counterproductive, rather than useful.

CONCLUSION

32. SQL injection attacks represent real threat, allowing the perpetrators to conduct attacks, which may lead to serious compromise, which could impact negatively on the business. The adoption of good security practice, such as the mitigation methods above, and an understanding of the business needs of the systems can lead to the reduction or elimination of SQL injection vulnerabilities.

REFERENCES

Database Providers

SQL injection is a well-known vulnerability and there is a considerable amount of information on all the providers' web sites. Some examples are:

<http://www.oracle.com>

<http://www.microsoft.com/sql/default.asp>

Risk assessment methodology

References for understanding the risk assessment methodology are:

http://www.theirm.org/publications/documents/Risk_Management_Standard_03_0820.pdf Risk Management Standard jointly published by AIRMIC (The

Association of Insurance and Risk managers), ALARM (The National Forum for Risk Management in the Public Sector) and IRM (Institute of Risk Management).

<http://www.itap.purdue.edu/security/assessments/> Principally for their own use, but it provides some useful tools.

http://www.dti.gov.uk/bestpractice/assets/security/risk_management.txt Offers guidance from the DTI on risk management in their best practices section.

<http://www.iso.org> ISO Guide 73 (Available for purchase) is a useful guide to the terminology and concepts of a risk management process.

White Papers

White papers provide detailed information into SQL injection attacks and how these attacks can be mitigated: Some examples are:

<http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf> A discussion of SQL injection in the context of web applications.

http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf A discussion of techniques to retrieve data from a database in the absence of error messages.

http://www.nextgenss.com/papers/advanced_sql_injection.pdf Discusses SQL injection techniques in the context of Microsoft IIS, Active Server Pages, and SQL Server.

http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf Provides advice on some more advanced SQL Injection attack techniques, including the use of time-delays for information gathering attacks.

<http://www.sqlsecurity.com/DesktopDefault.aspx?tabid=24> A reasonably comprehensive SQL Server Lockdown checklist. Also, check the SQLSecurity site's Doc/FAQ section for many helpful SQL Server documents.

<http://www.securityfocus.com/infocus/1644> SQL Injection and Oracle, part 1.

<http://www.securityfocus.com/infocus/1646> SQL Injection And Oracle, parts 2.

<http://www.petefinnigan.com/orasec.htm> Oracle lockdown information.