Database Security

Brendan Tierney

1

Ongoing Meow attack has nuked >1,000 databases without telling anyone why

Ongoing attack hitting unsecured data leaves the word "meow" as its calling card.

DAN GOODIN - 7/22/2020, 11:42 PM

2

Database Security

- Data is a valuable resource that must be strictly controlled and managed, as with any corporate resource.
- Part or all of the corporate data may have strategic importance and therefore needs to be kept secure and confidential.
- Mechanisms that protect the database against intentional or accidental threats.
- Security considerations do not only apply to the data held in a database. Breaches of security may affect other parts of the system, which may in turn affect the database.

3

Security Risks

- External threats:
  - Unauthorized users
  - Denial of service
  - Unauthorized data access
  - Exploits: SQL injection and others
- Internal threats:
  - Abuse: Data theft
  - Sabotage: Data or service corruption
  - Complexity
  - Recovery
  - Omission
  - External threats listed above
- Partners

4

Security

- **Security** - protection from malicious attempts to steal or modify data.
  - Database system level
    - Authentication and authorization mechanisms to allow specific users access only to required data
  - Operating system level
    - Operating system super-users can do anything they want to the database!
    - Good operating system level security is required.
  - Network level: must use encryption to prevent
    - Eavesdropping (unauthorized reading of messages)
    - Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)
  - Physical level
    - Physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed
    - Computers must also be protected from floods, fire, etc.
  - Human level
    - Users must be screened to ensure that an authorized users do not give access to intruders
    - Users should be trained on password selection and secrecy

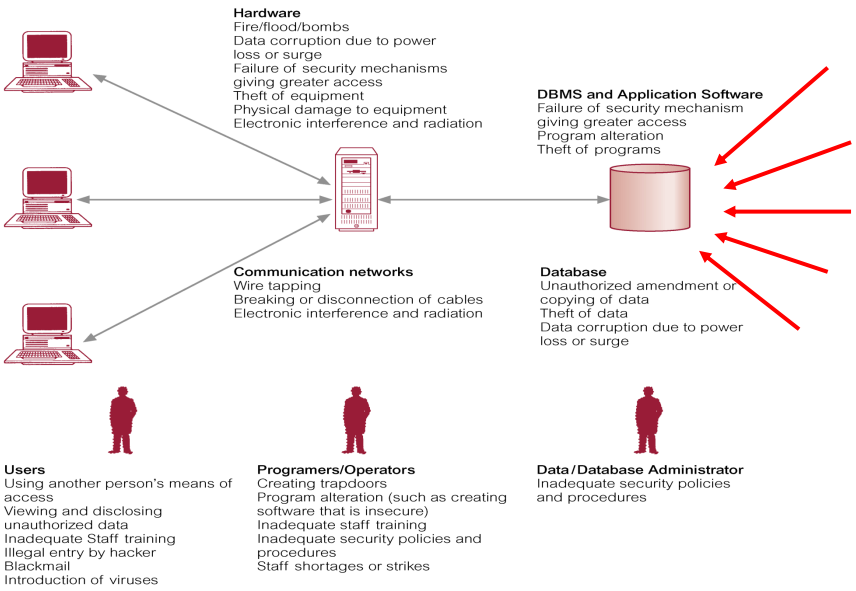
5

Threats

- Loss of Integrity
  - Protect from improper modification
  - Intentional or accident
- Loss of Availability
  - Major issue in 24x7 environments
- Loss of Confidentiality
  - Need to protect the data from unauthorised and accidental disclosure
- Main database counter measure is access control

6

Threats



7

Introduction to DB Security

- In a multi-user environment users should access selected portions of the data
  - 3 schema architecture is your starting point
  - Need to protect sensitive information e.g. salaries
- **Secrecy**: Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students' grades.
- **Integrity**: Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
    - Teaching assistants should not be able to modify but be able to view
- **Availability**: Users should be able to see and modify things they are allowed to.

8

Principle of Least Privilege

- Install only the required software on the machine.
- Activate only the required services on the machine.
- Give operating system (OS) and database access to only those users who require access.
- Limit access to the root or administrator account.
- Limit access to privileged database accounts.
- Limit users' access to only the database objects that they require to do their jobs.

9

Access Controls

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access

- A **security policy** specifies who is authorized to do what.
- A **security mechanism** allows us to enforce a chosen security policy.

- **Authentication**
  - A mechanism that determines whether a user is, who he or she claims to be.
- **Authorisation**
  - The granting of a right or privilege that enables a subject to have legitimate access to a database system or a database system's object.
    - Authorization determines the user's privileges.
      - Privileges
      - Roles

- Two main mechanisms at the DBMS level:
  - Discretionary access control
    - Granting of privileges to users
  - Mandatory access control
    - Access based security level of user via roles

10

Database Administrator

- Has overall responsibility of implementing database security
  - Data security
  - DBMS security
  - Server
- A new user
  - Create an account with password
    - But can very few operations
  - Assign space allocations and locations
  - Assign access privileges
    - Default is CONNECT
  - Assign roles
    - dt2284

11

Database Security

- The database system must also keep **track of all operations** on the database that are applied by a certain user throughout **each login session**.
  - To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify **system log**, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.
- If any tampering with the database is suspected, a **database audit** is performed
  - A database audit consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.

12

Authorization

Forms of authorization on parts of the database:

- **Read authorization** - allows reading, but not modification of data.
- **Insert authorization** - allows insertion of new data, but not modification of existing data.
- **Update authorization** - allows modification, but not deletion of data.
- **Delete authorization** - allows deletion of data

Forms of authorization to modify the database schema:

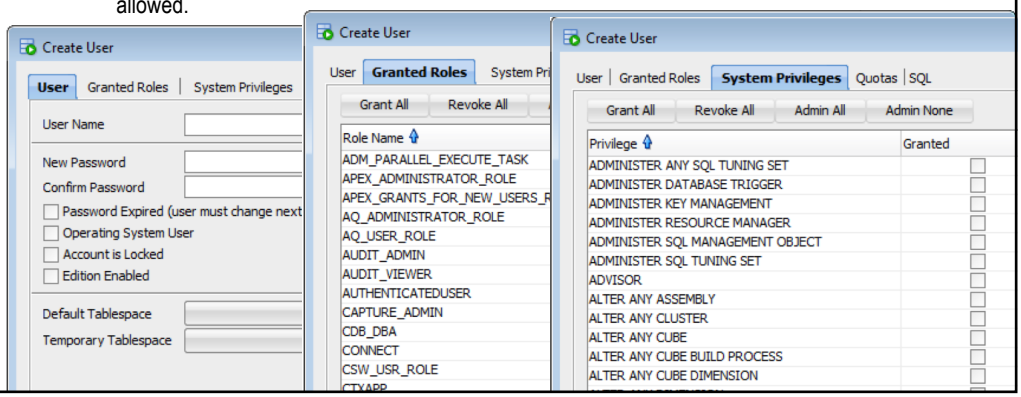
- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations

13

Discretionary Access Control

- Based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).

- Creator of a table or a view automatically gets all privileges on it.
  - DMBS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.



14

Types of Discretionary Privileges

- The **account level**:
  - At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- The **relation level** (or **table level**):
  - At this level, the DBA can control the privilege to access each individual relation or view in the database.

15

Types of Discretionary Privileges

- The privileges at the **account level** apply to the capabilities provided to the account itself and can include
  - the **CREATE SCHEMA** or **CREATE TABLE** privilege, to create a schema or base relation;
  - the **CREATE VIEW** privilege;
  - the **ALTER** privilege, to apply schema changes such adding or removing attributes from relations;
  - the **DROP** privilege, to delete relations or views;
  - the **MODIFY** privilege, to insert, delete, or update tuples;
  - and the **SELECT** privilege, to retrieve information from the database by using a **SELECT** query
  - ……

16

Types of Discretionary Privileges

- The second level of privileges applies to the **relation level**
  - This includes **base relations** and virtual (**view**) relations.
- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model where
  - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
  - The **columns** represent **objects** (relations, records, columns, views, operations).
  - Each position **M(i,j)** in the matrix represents the types of privileges (read, write, update) that **subject i** holds on **object j**.

17

Types of Discretionary Privileges

- To control the granting and revoking of relation privileges, each relation R in a database is assigned an **owner account**, which is typically the account that was used when the relation was created in the first place.
  - The owner of a relation is given all privileges on that relation.
  - The DBA can assign and owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the **CREATE SCHEMA** command.
  - The owner account holder can **pass privileges** on any of the owned relation to other users by **granting** privileges to their accounts.

18

Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *branch* relation:

    grant select on *branch* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges

19

Security Specification in SQL

- The grant statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name> to <user list>
- <user list> is:
  - a user-id
  - *public*, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

20

Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
  - Example:

    grant select on *branch* **to** U_1 **with grant option**

    gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others

21

GRANT Command

GRANT privileges ON object TO users [WITH GRANT OPTION]

- The following privileges can be specified:
  - SELECT: Can read all columns (including those added later via ALTER TABLE command).
  - INSERT(col-name): Can insert tuples with non-null or non-default values in this column.
    - INSERT means same right with respect to all columns.
  - DELETE: Can delete tuples.
  - REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.
- If a user has a privilege with the GRANT OPTION, can pass privilege on to other users (with or without passing on the GRANT OPTION).
  - Grant propagation
- Only owner can execute CREATE, ALTER, and DROP.

22

GRANT and REVOKE of Privileges

- GRANT INSERT, SELECT ON EMPLOYEES TO BRENDAN
  - BRENDAN can query EMPLOYEES or insert tuples into it.
- GRANT DELETE ON EMPLOYEES TO SEAN WITH GRANT OPTION
  - SEAN can delete tuples, and also authorize others to do so.
- GRANT UPDATE ON EMPLOYEE (SALARY) TO DECLAN
  - DECLAN can only update the *salary* field of EMPLOYEES.

- REVOKE: When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.

23

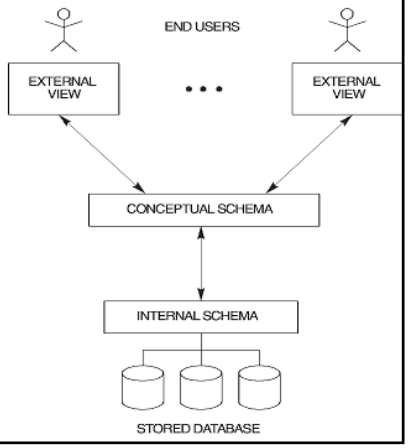
Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).

- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.

- Together with GRANT/REVOKE commands, views are a very powerful access control tool.

- Example



24

Authorisation and Views

- Users can be given authorization on views, without being given any authorisation on the relations used in the view definition
- Ability of views to hide data serves both to simplify usage of the system and to enhance security by allowing users access only to data they need for their job
- A combination or relational-level security and view-level security can be used to limit a user's access to precisely the data that user needs.

25

View Example

- Suppose a bank clerk needs to know the names of the customers of each branch, but is not authorized to see specific loan information.
  - Approach: Deny direct access to the *loan* relation, but grant access to the view *cust-loan*, which consists only of the names of customers and the branches at which they have a loan.
  - The *cust-loan* view is defined in SQL as follows:

```
create view cust-loan as
    select branchname, customer-name
    from   borrower, loan
    where borrower.loan-number = loan.loan-number
```

26

View Example (Cont.)

- The clerk is authorized to see the result of the query:

```
select *
from cust-loan
```

- When the query processor translates the result into a query on the actual relations in the database, we obtain a query on *borrower* and *loan*.
- Authorization must be checked on the clerk's query before query processing replaces a view by the definition of the view.

27

Authorization on Views

- Creation of view does not require **resources** authorization since no real relation is being created
- The creator of a view gets only those privileges that provide no additional authorization beyond that he already had.
- E.g. if creator of view *cust-loan* had only **read** authorization on *borrower* and *loan*, he gets only **read** authorization on *cust-loan*

28

Granting of Privileges

- The passage of authorization from one user to another may be represented by an authorization graph.
- The nodes of this graph are the users.
- The root of the graph is the database administrator.
- Consider graph for update authorization on loan.
- An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on loan to U_j.

29

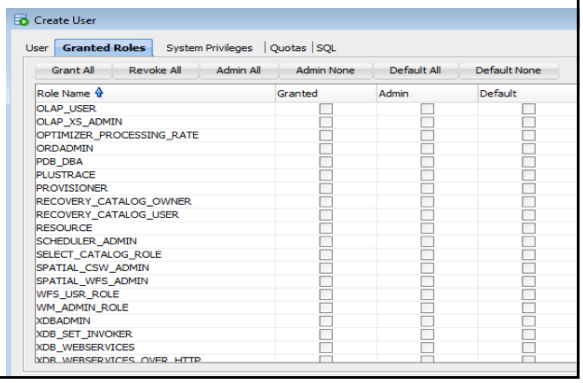
Authorization Grant Graph

- *Requirement*: All edges in an authorization graph must be part of some path originating with the database administrator
- If DBA revokes grant from U_1:
  - Grant must be revoked from U_4 since U_1 no longer has authorization
  - Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U_7
  - U7 grants authorization to U_8
  - U8 grants authorization to U_7
  - DBA revokes authorization from U_7
- Must revoke grant U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore.

30

Role-Based Authorization

- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Based on notion the permissions are associated with roles and users are assigned to appropriate roles
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.
  - Will have 1 or more privileges



31

Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding "role"
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports roles

create role *teller*
create role *manager*

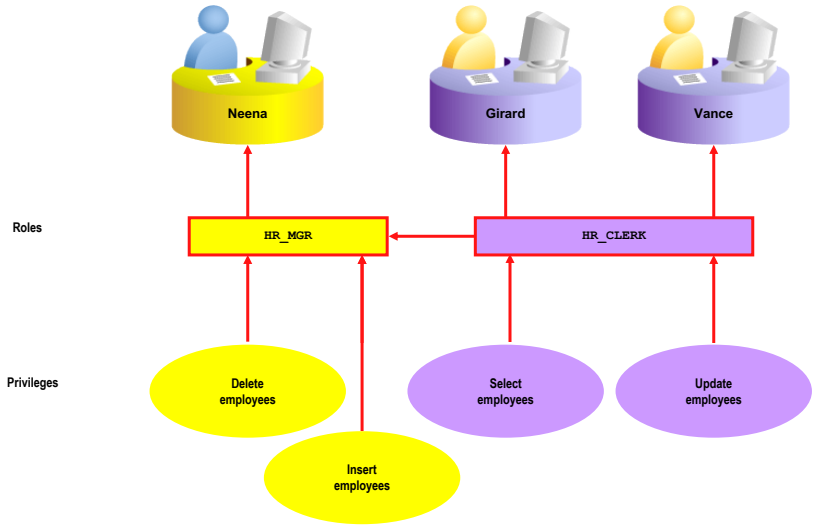
grant select on *branch* **to** *teller*
grant update (*balance***) on** *account* **to** *teller*
grant all privileges on *account* **to** *manager*

grant *teller* **to** *manager*

grant *teller* **to** *alice, bob*
grant *manager* **to** *avi*

32

Roles



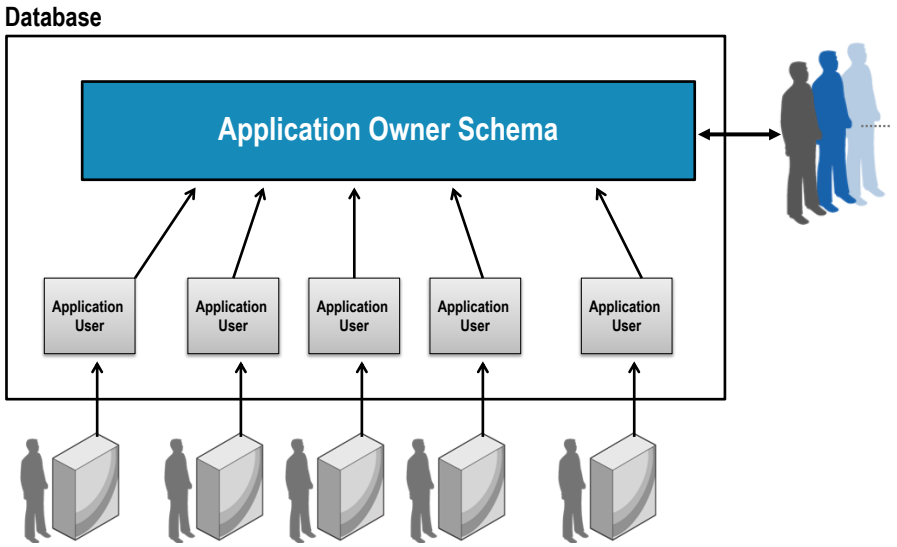
33

Predefined Roles

CONNECT	CREATE SESSION
RESOURCE	CREATE TABLE, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TRIGGER, CREATE TYPE, CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR
SCHEDULER_ADMIN	CREATE ANY JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
DBA	Most system privileges, several other roles. Do not grant to nonadministrators.

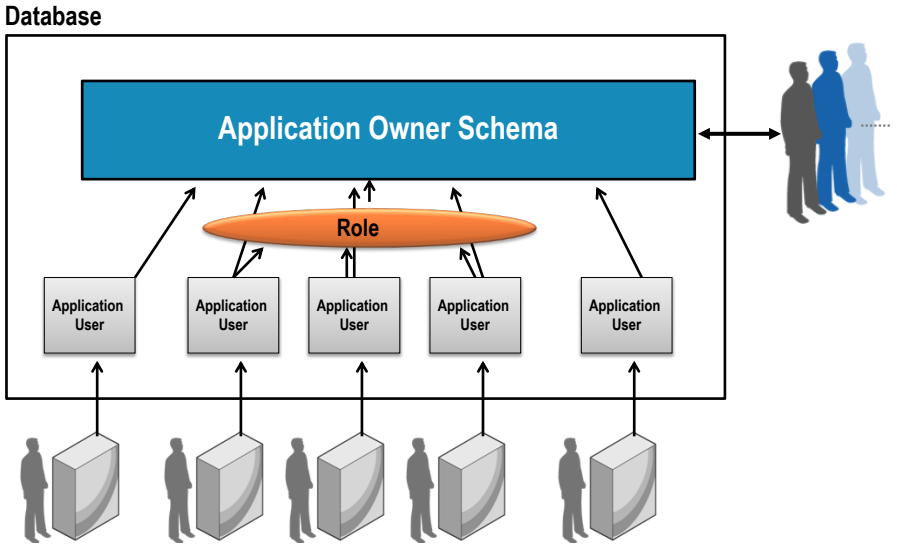
34

Schema Owner & Application User Grants (C/S)



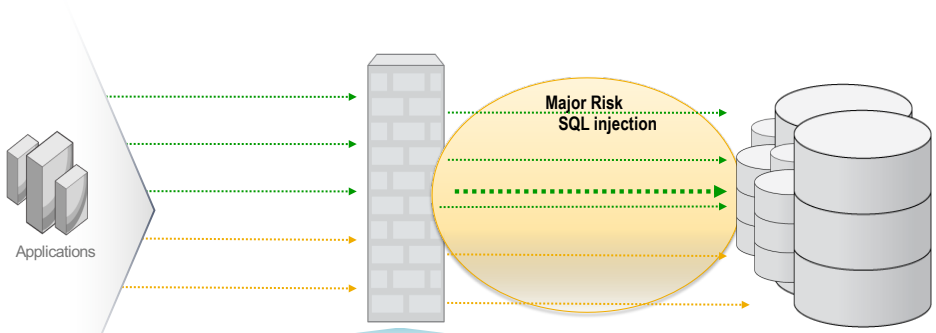
35

Schema Owner & Application User Grants – Using Roles (C/S)



36

Typical 3 Tier – plus a bit more



What should we have instead ?

Yes it is a lot more work
But a lot more secure

37

Security

- Defaults are not changed
- Can have limited security features compared to longer standing DBs
- Some have hard coded “admin” passwords
  - Some insist on passwords being changed on install. Some don’t.

Security

MongoDB ransom attacks soar, body count hits 27,000 in hours

Aussie comms watchdog reporting exposed databases.

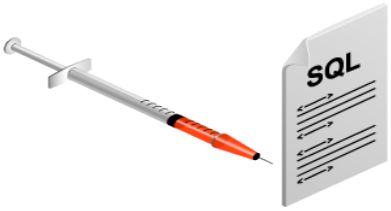
9 Jan 2017 at 02:26, Darren Pauli

MongoDB databases are being decimated in soaring ransomware attacks that have seen the number of compromised systems more than double to 27,000 in a day.

38

Understanding SQL Injection

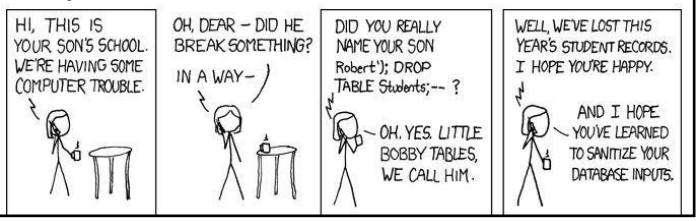
- SQL injection is a technique for maliciously exploiting applications that use client-supplied data in SQL statements.
  - Attackers trick the SQL engine into executing unintended commands.
  - SQL injection techniques may differ, but they all exploit a single vulnerability in the application.
  - To immunize your code against SQL injection attacks, use bind arguments or validate and sanitize all input concatenated to dynamic SQL.



39

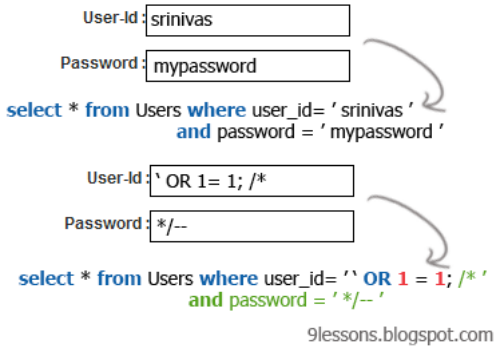
Preventing SQL Injection

- SQL injection can be prevented by:
  - Reducing the attack surface
    - Use invoker's rights.
    - Reduce arbitrary inputs.
  - Avoiding dynamic SQL with concatenated input
    - Use static SQL.
    - Use bind arguments.
    - Validate input.
  - Designing code that is immune to SQL injections
  - Testing code for SQL injection flaws



40

SQL Injection.



41

SQL Injection: Example

```
-- First order attack
CREATE OR REPLACE PROCEDURE GET_EMAIL
  (p_last_name VARCHAR2 DEFAULT NULL)
AS
  TYPE    cv_custtyp IS REF CURSOR;
  cv      cv_custtyp;
  v_email customers.cust_email%TYPE;
  v_stmt  VARCHAR2(400);
BEGIN
  v_stmt := 'SELECT cust_email FROM customers
            WHERE cust_last_name = '''|| p_last_name || '''';
  DBMS_OUTPUT.PUT_LINE('SQL statement: ' || v_stmt);
  OPEN cv FOR v_stmt;
  LOOP
    FETCH cv INTO v_email;
    EXIT WHEN cv%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Email: '||v_email);
  END LOOP;
  CLOSE cv;
EXCEPTION WHEN OTHERS THEN
  dbms_output.PUT_LINE(sqlerrm);
  dbms_output.PUT_LINE('SQL statement: ' || v_stmt);
END;
```

String literals that are incorrectly validated or not validated are concatenated into a dynamic SQL statement, and interpreted as code by the SQL engine.

42

SQL Injection: Example

- The example on previous slide demonstrates a procedure with dynamic SQL constructed via concatenation of input value. This is vulnerable to SQL injection

```
SQL> EXECUTE get_email('Andrews')
SQL statement: SELECT cust_email FROM customers WHERE cust_last_name = 'Andrews'
Email: Ajay.Andrews@YELLOWTHROAT.COM
Email: Dianne.Andrews@TURNSTONE.COM

PL/SQL procedure successfully completed.

SQL> EXECUTE get_email('x'' union select username from all_users where ''x''=''x')
SQL statement: SELECT cust_email FROM customers WHERE cust_last_name = 'x' union
select username from all_users where 'x'='x'
Email: ANONYMOUS
Email: APEX_PUBLIC_USER
Email: BI
Email: CTXSYS
...
```

43

Protecting Against SQL Injection: Example

- This example in the slide with static SQL is protected against SQL injection.

```
CREATE OR REPLACE PROCEDURE GET_EMAIL
  (p_last_name VARCHAR2 DEFAULT NULL)
AS
BEGIN
  FOR i IN
    (SELECT cust_email
     FROM customers
     WHERE cust_last_name = p_last_name)
  LOOP
    DBMS_OUTPUT.PUT_LINE('Email: '||i.cust_email);
  END LOOP;
END;
```

This example avoids dynamic SQL with concatenated input values.

```
EXECUTE get_email('Andrews')
Email: Ajay.Andrews@YELLOWTHROAT.COM
Email: Dianne.Andrews@TURNSTONE.COM

PL/SQL procedure successfully completed.

EXECUTE get_email('x'' union select username from all_users where ''x''=''x')

PL/SQL procedure successfully completed.
```

44

Using Invoker's Rights

- OE is successful at changing the SYS password, because, by default, CHANGE_PASSWORD executes with SYS privileges:

```
CONNECT oe

EXECUTE sys.change_password ('SYS', 'mine')

PL/SQL procedure successfully completed.
```

- Add the AUTHID to change the privileges to the invokers:

```
CONNECT /as sysdba
CREATE OR REPLACE
PROCEDURE change_password(p_username VARCHAR2 DEFAULT NULL,
                          p_new_password VARCHAR2 DEFAULT NULL)
AUTHID CURRENT_USER
IS
  v_sql_stmt VARCHAR2(500);
BEGIN
  v_sql_stmt := 'ALTER USER '||p_username ||' IDENTIFIED BY '
                || p_new_password;
  EXECUTE IMMEDIATE v_sql_stmt;
END change_password;
```

45

Using Invoker's Rights

- When OE executes the CHANGE_PASSWORD procedure, it is executed under SYS privileges (the definer of the procedure), and with the code shown, OE can change the SYS password. Obviously, this is an unacceptable outcome.
- To disallow another schema from changing a password that does not belong to the schema, redefine the procedure with the invoker's rights. This is done with the AUTHID CURRENT_USER option.

```
CONNECT oe

EXECUTE change_password ('SYS', 'mine')

ERROR at line 1:
ORA-01031: Insufficient privileges
ORA-06512: at "SYS.CHANGE_PASSWORD", at line 1
ORA-06512: at line 1
```

- Now OE can no longer change the SYS (or any other account) password.
- Notice that the CHANGE_PASSWORD procedure contains dynamic SQL with concatenated input values. This is a SQL injection vulnerability. Although using invoker's rights does not guarantee the elimination of SQL injection risks, it can help mitigate the exposure.

46

Avoiding SQL Injection - Using Static SQL

```
CREATE OR REPLACE PROCEDURE list_products_dynamic
(p_product_name VARCHAR2 DEFAULT NULL)
AS
  TYPE cv_prodtyp IS REF CURSOR;
  cv   cv_prodtyp;
  v_prodname product_information.product_name%TYPE;
  v_minprice product_information.min_price%TYPE;
  v_listprice product_information.list_price%TYPE
  v_stmt  VARCHAR2(400);
BEGIN
  v_stmt := 'SELECT product_name, min_price, list_price
             FROM product_information WHERE product_name LIKE
             ''%''||p_product_name||''%''';
  OPEN cv FOR v_stmt;
  dbms_output.put_line(v_stmt);
  LOOP
    FETCH cv INTO v_prodname, v_minprice, v_listprice;
    EXIT WHEN cv%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Product Info: '||v_prodname ||', '||
                         v_minprice ||', '|| v_listprice);
  END LOOP;
  CLOSE cv;
END;
```

You can convert this statement to static SQL.

47

Using Static SQL

- To use static SQL, accept the user input, and then concatenate the necessary string to a local variable.
- Pass the local variable to the static SQL statement.

```
CREATE OR REPLACE PROCEDURE list_products_static
   (p_product_name VARCHAR2 DEFAULT NULL)
AS
  v_bind  VARCHAR2(400);
BEGIN
  v_bind := '%'||p_product_name||'%';
  FOR i in
    (SELECT product_name, min_price, list_price
     FROM product_information
     WHERE product_name like v_bind)
  LOOP
      DBMS_OUTPUT.PUT_LINE('Product Info: '||i.product_name ||',
         '|| i.min_price ||', '|| i.list_price);
  END LOOP;
END list_products_static;
```

48

Using Static SQL

- To use static SQL, accept the user input, and then concatenate the necessary string to a local variable.
- Pass the local variable to the static SQL statement.

Examine the results:

```
-- desired results – normal execution
EXECUTE list_products_static('Laptop')
Product Info: Laptop 128/12/56/v90/110, 2606, 3219
Product Info: Laptop 16/8/110, 800, 999
Product Info: Laptop 32/10/56, 1542, 1749
Product Info: Laptop 48/10/56/110, 2073, 2556
Product Info: Laptop 64/10/56/220, 2275, 2768

PL/SQL procedure successfully completed.

-- this example attempts injection
EXECUTE list_products_static(''' and 1=0 union select cast(username as
varchar2(100)), null, null from all_users --')

PL/SQL procedure successfully completed.
```

49

Using Dynamic SQL

- Dynamic SQL may be unavoidable in the following types of situations:
  - You do not know the full text of the SQL statements that must be executed in a PL/SQL procedure.
  - You want to execute DDL statements and other SQL statements that are not supported in purely static SQL programs.
  - You want to write a program that can handle changes in data definitions without the need to recompile.
- If you must use dynamic SQL, try not to construct it through concatenation of input values. Instead, use bind arguments.

50

Using Bind Arguments with Dynamic SQL

- You can rewrite the following statement

```
v_stmt :=
 'SELECT '||filter(p_column_list)||' FROM customers '||
 'WHERE account_mgr_id = '''|| p_sales_rep_id ||'''';

EXECUTE IMMEDIATE v_stmt;
```

- as this dynamic SQL with a placeholder (`:1`) by using a bind argument (`p_sales_rep_id`):

```
v_stmt :=
 'SELECT '||filter(p_column_list)||' FROM customers '||
 'WHERE account_mgr_id = :1';

EXECUTE IMMEIDATE v_stmt USING p_sales_rep_id;
```

51

Using Bind Arguments with Dynamic PL/SQL

- If you must use dynamic PL/SQL, try to use bind arguments. For example, you can rewrite the following dynamic PL/SQL with concatenated string values:

```
v_stmt :=
  'BEGIN
     get_phone (''' || p_fname ||
                ''','''|| p_lname ||'''); END;';

EXECUTE IMMEDIATE v_stmt;
```

- as this dynamic PL/SQL with placeholders (`:1, :2`) by using bind arguments (`p_fname, p_lname`):

```
v_stmt :=
  'BEGIN
     get_phone(:1, :2); END;';

EXECUTE IMMEIDATE v_stmt USING p_fname, p_lname;
```

52

Reducing the Attack Surface

- Using invoker's rights helps to:
  - Limit the privileges
  - Minimize the security exposure

```
-- this example does not use Invoker's rights
CONNECT / as sysdba

CREATE OR REPLACE
PROCEDURE change_password(p_username VARCHAR2 DEFAULT NULL,
                          p_new_password VARCHAR2 DEFAULT NULL)
IS
  v_sql_stmt VARCHAR2(500);
BEGIN
  v_sql_stmt := 'ALTER USER '||p_username ||' IDENTIFIED BY '
                || p_new_password;
  EXECUTE IMMEDIATE v_sql_stmt;
END change_password;
```

Note the use of dynamic SQL with concatenated input values.

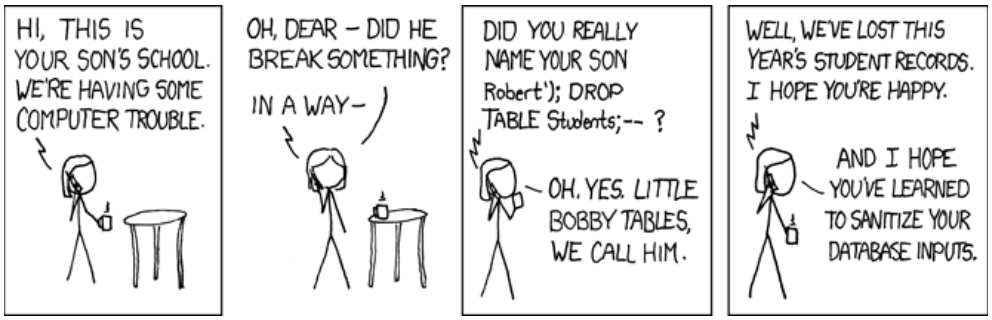
1

```
GRANT EXECUTE ON change_password to OE, HR, SH;
```

2

53

54



55

Transaction Monitoring

- Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly whenever a table or view is modified or when some user actions or database system actions occur.
  - Fires whenever one of the following operations occurs:
    - DML statements on a particular schema object,
    - DDL statements issued within a schema or database,
    - user logon or logoff events,
    - server errors,
    - database startup,
    - instance shutdown
- Triggers are run when a triggering event occurs, no matter which user is connected or which application is being used

56

Transaction Monitoring

- Triggers can run either before or after an event
- Typical events include INSERT, UPDATE, DELETE
  - Row level triggers
  - Others include CREATE, ALTER, DROP
    - Schema level triggers

```
CREATE OR REPLACE TRIGGER Log_employee_changes
BEFORE UPDATE ON    EMPLOYEE
FOR EACH ROW
BEGIN
  -- copy of the old data before the update
  INSERT INTO Emp_log (Emp_id, Log_date, New_salary, Action) VALUES
  (:old.Empno, SYSDATE, :old.SAL, 'Old Details');
  -- copy of new data after the update
  INSERT INTO Emp_log (Emp_id, Log_date, New_salary, Action) VALUES
  (:new.Empno, SYSDATE, :new.SAL, 'New Details');
END;
```

57

Problem

- Discretionary control has some flaws, e.g., the *Trojan horse* problem:
  - John creates Horsie and gives INSERT privileges to Justin (who doesn't know about this).
  - John modifes the code of an application program used by Justin to additionally write some secret data to table Horsie.
  - Now, Justin can see the secret info.
- The modification of the code is beyond the DBMSs control, but it can try and prevent the use of the database as a channel for secret information

58

Mandatory Access Control

- Discretionary access control is a all or nothing method
  - You either have or do not have a certain privilege
- Mandatory Access Control is based the idea of having an additional security policy, where data and users are classified based on security classes
  - Typical application areas are military, government and intelligence agencies
  - Each object is assigned a security class, each user is assigned clearance for a security class, and rules are imposed on reading and writing of objects by users
- Typically Discretionary & Mandatory Access Control are combined in some way
  - Commercial DBs only have discretionary

59

Mandatory Access Control

- Classes include
  - Top Secret (TS), Secret (S), Confidential (c), Unclassified (U)
  - TS ≥ S ≥ C ≥ U
- Commonly used model for multilevel security classifies each subject and object into one of the security classifications
  - Subject (user, account, program)
  - Object (relation, tuple, column, view, operation)
- Restrictions
  - A Subject is not allowed read access to an object O unless class(S) ≥ class(O)
  - A Subject is not allowed write an object O unless class(S) ≤ class(0)

60

Comparison

- Discretionary Access Control
  - High degree of flexibility
  - Suitable for a large variety of applications
  - Drawback is their vulnerability to malicious attacks
- Mandatory Access Control
  - Ensure high degree of protection
  - Very rigid
  - Prevent illegal flow of information
  - Suitable for a small number of applications

61

Security in Distributed Databases

- Integrity & Security maintenance are important when systems are physically separated
  - There is a need to interpret local integrity & security requirements in a global environment
- Data is stored & managed locally & remotely
  - The representations, modelling and management of remote data may differ from those of local sites.
- In a DDBMS mechanisms exist for ensuring that only authorised users can perform operation on the data
  - Those operations must be correct with respect to the data definition and semantics
- The diversity of possible security definitions make it extremely difficult to specify security mechanisms that can be applied in a DDBMS

62

Security in Distributed Databases

- Security management & mechanisms are straight forward in centralised DBMS
  - Single site holds one copy of all the required security tables in the system catalogue
  - All access attempts are checked against a single copy of the security rules
  - Therefore a centralised DBMS has all the information needed to secure its data
- In a DDBMS there are 2 basis approaches to security management
  - Offer local autonomy to each site within the system
  - Centralise the security to a central site that defines and enforces policy for the entire distributed system

63

Security in Distributed Databases

- Security Problems
  - How do you manage non-local users
  - 3 ways (4 ways)
    - Local system requires copies of the authentication information from the remote site where the user was authenticated . This information is integrated into the local security data for the duration of the user's session
    - Local system accepts that the remote system has authenticated the user and accepts the user's requests without further local security enforcement
    - Authority descriptions from all local sites can be replicated and distributed throughout the entire system.
    - ? – is there a better way
- One common approach is the use of roles as the basis of defining access rights

64

Security in Distributed Databases

- In a DDBMS the mechanisms for implementing and enforcing permissions may differ
  - Using the categorisation of users by roles should make this uniform
  - A user is associated with a role, which is defined globally. The role identifier is transmitted with each request at a remote site.
  - The receiving site enforces local security checks appropriate for that role
  - In a centralised mechanism all sites are dependent on the central security mechanism.
    - Need to decide if there will be a single copy of the security information
      - Performance bottleneck as one site does all checking
      - Single point of failure
    - Or it is replicated to all sites
      - Copies are several different sites
      - Full or partial replications

65

Security in Distributed Databases

- DDBMS Security summary
  - Presence of multiple data access points
  - Need to ensure adequate security is enforced at all sites
  - The weakest security within the DDBMS determines the overall level of security
  - Replication of data to improve performance must be controlled to ensure appropriate access to levels of data
  - It should be possible to ensure that unauthorised access at one site does not result in overall access to the global system

66

67

What about securing your Business Logic ?

(that is stored in the database)

Discuss ……..

Multi-level approach to securing your data and code in the database

68

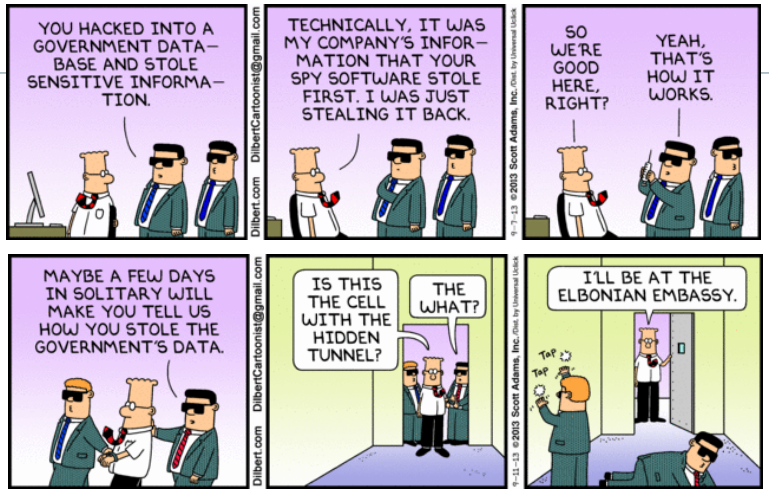
Discuss

A user should only have the necessary privileges to execute a command, for only during the time of the execution of the command.

Is this a good idea or a bad idea?

How would you implement this?

69



70

Exercises

- Using the Oracle VM

- Log into the PDB as SYS/SYSTEM
- Create a new schema called TEACHER (Temp=Temp, Default=USER)
- Grant Connect & Resource privileges
- Try connecting to the TEACHER schema
- Create a table called STUDENT
- Insert 10 record into the table

- Create another schema called STUDENT001
- Grant Connect privilege to the STUDENT001 schema
- Connect to STUDENT001 schema & create a table called ASSIGNMENT

71

Exercises

- Create a Role called STUDENT_ACCESS
- Grant the Select privileges to the STUDENTS table to this role
- Grant the role to STUDENT001

- Connect as STUDENT001
- Create a table called assignment
- Query the STUDENT table

- What can you do next?

- Grant Update on the table to the role?
- Test that this works? How can you do that?

72