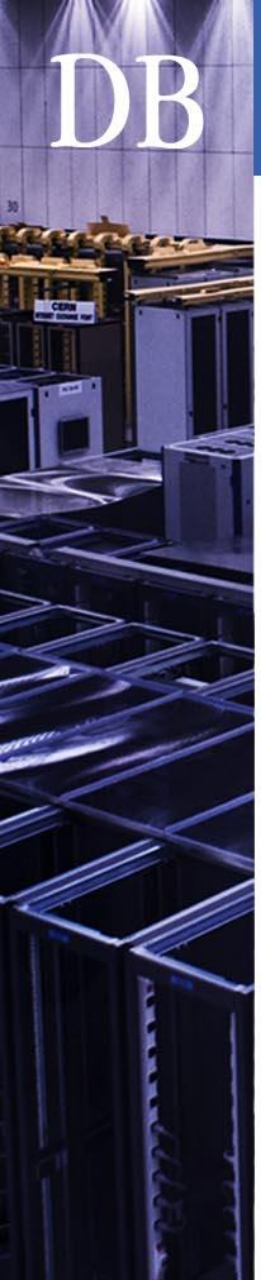


Database security tutorial

Part I

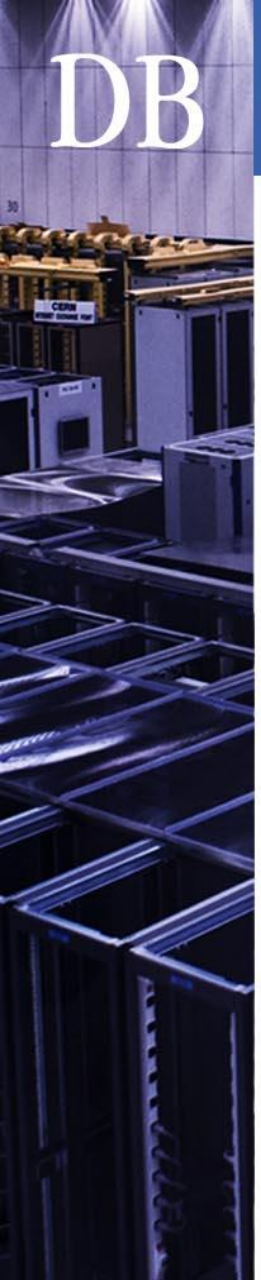
Oracle Tutorials, June 4th 2012

Daniel Gómez Blanco

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Database SERVICES

CERN



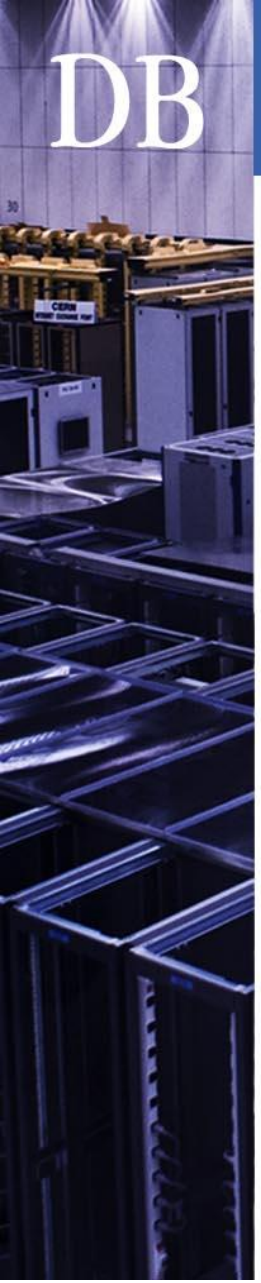
Agenda

- Authentication
- Roles and privileges
- Auditing

Authentication

- Basis of any security model
- Process of confirming the correctness of the claimed identity
- User account
  - Unique username (associated schema)
  - Authentication method (password, global or external)
  - Default and temporary tablespaces
  - User profile
  - Account status
- Predefined administrative accounts
  - `SYS`
  - `SYSTEM`
  - `DBSNMP`
  - `SYSMAN`
- Least privilege principle
- Syntax

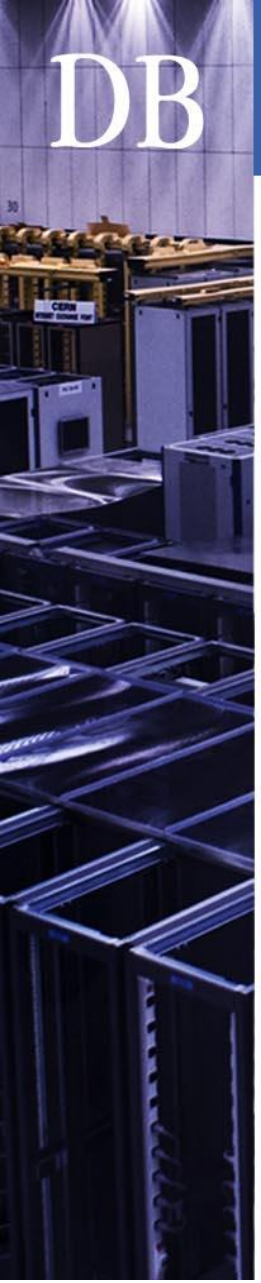
```
CREATE USER username IDENTIFIED BY user_password
    DEFAULT TABLESPACE tblspc QUOTA 10M ON tblspc
    TEMPORARY TABLESPACE tmpspc QUOTA 5M on tmpspc
    PROFILE sec_prof PASSWORD EXPIRE;
```



Authentication: passwords

- Often far too easy to crack (default, empty, simple)
- Do
  - Use mixed-case letters
  - Use numbers
  - Use punctuation marks
  - At least 8 characters
- Don't
  - Use same password all over the place
  - Use username as password or any permutation
  - Use dictionary words
  - Use easily obtained information
  - Use dates
- Password checking tools
  - Oracle Auditing Tool (OAT) – Oracle Password Guesser
    - http://www.vulnerabilityassessment.co.uk/oat.htm
  - Oracle Password Cracker by Pete Finnigan
    - http://www.petefinnigan.com/oracle_password_cracker.htm
- Enforce use of password profiles and account lockouts

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



Authentication: profiles

- Profiles impose limits on DB usage and manage account status and password management rules
- Account locking
  - FAILED_LOGIN_ATTEMPS
  - PASSWORD_LOCK_TIME
- Password aging and expiration
  - PASSWORD_LIFE_TIME
  - PASSWORD_GRACE_TIME
- Password history
  - PASSWORD_REUSE_TIME
  - PASSWORD_REUSE_MAX
- Password complexity verification
  - PASSWORD_VERIFY_FUNCTION
- Syntax

```
CREATE PROFILE sec_prof LIMIT
    FAILED_LOGIN_ATTEMPS 5;
ALTER USER username PROFILE secprof;
```

Roles and privileges: privileges

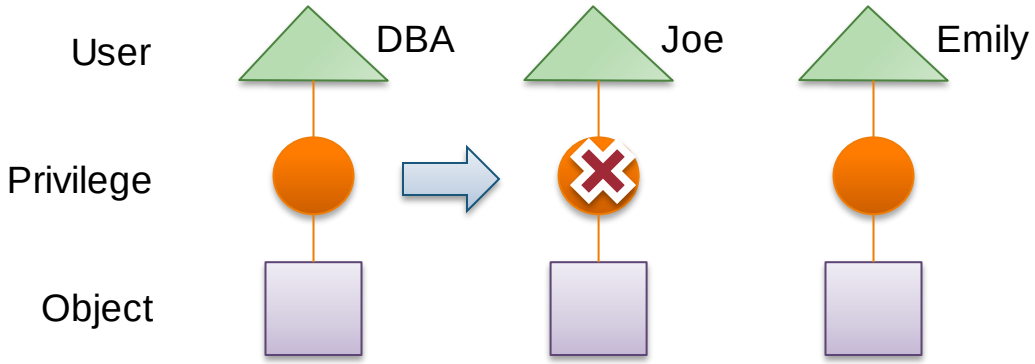
CERN IT Department

- A *privilege* is a right to execute a particular type of SQL statement or to access another user's object
- System privileges
  - Enable users to perform actions in the database
  - `GRANT <system_privilege> TO <grantee_clause> [WITH ADMIN OPTION]`
  - `REVOKE <system_privilege> FROM <grantee_clause>`
- Object privileges
  - Enable users to access and manipulate a specific object
  - `GRANT <object_privilege> ON <object> TO <grantee_clause> [WITH GRANT OPTION]`
  - `REVOKE <object_privilege> FROM <grantee_clause>`

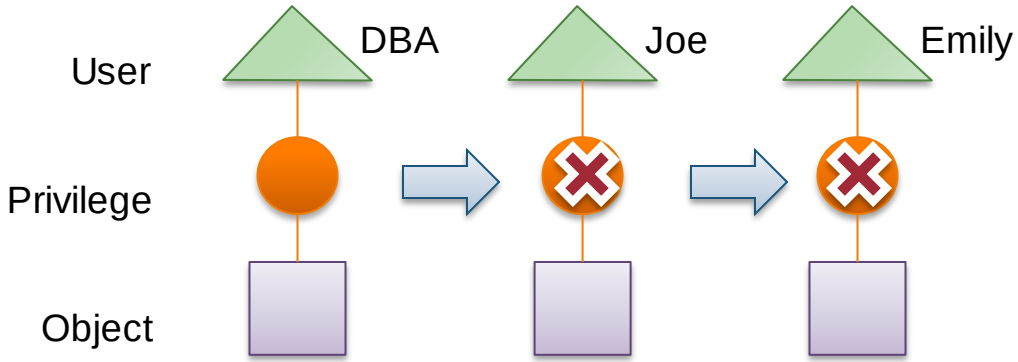
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Roles and privileges: revoking privileges

- Revoking system privilege



- Revoking object privileges



CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Roles and privileges: roles

CERN IT Department

- A *role* is a named group of related privileges that are granted to users or to other roles
- Privileges granted to and revoked from roles as users

```
GRANT <privilege> TO <role> [WITH ADMIN/GRANT OPTION]
```

- Roles granted to and revoked from users as system privileges

```
GRANT <role> TO <user> [WITH ADMIN OPTION]
```

- Predefined roles (`CONNECT, RESOURCE, SCHEDULER_ADMIN, DBA, SELECT_CATALOG_ROLE`)
- Roles can be secured
  - Set password to enable roles
  - Make roles non-default
  - Create roles than can be enabled only through PL/SQL procedures
- Syntax

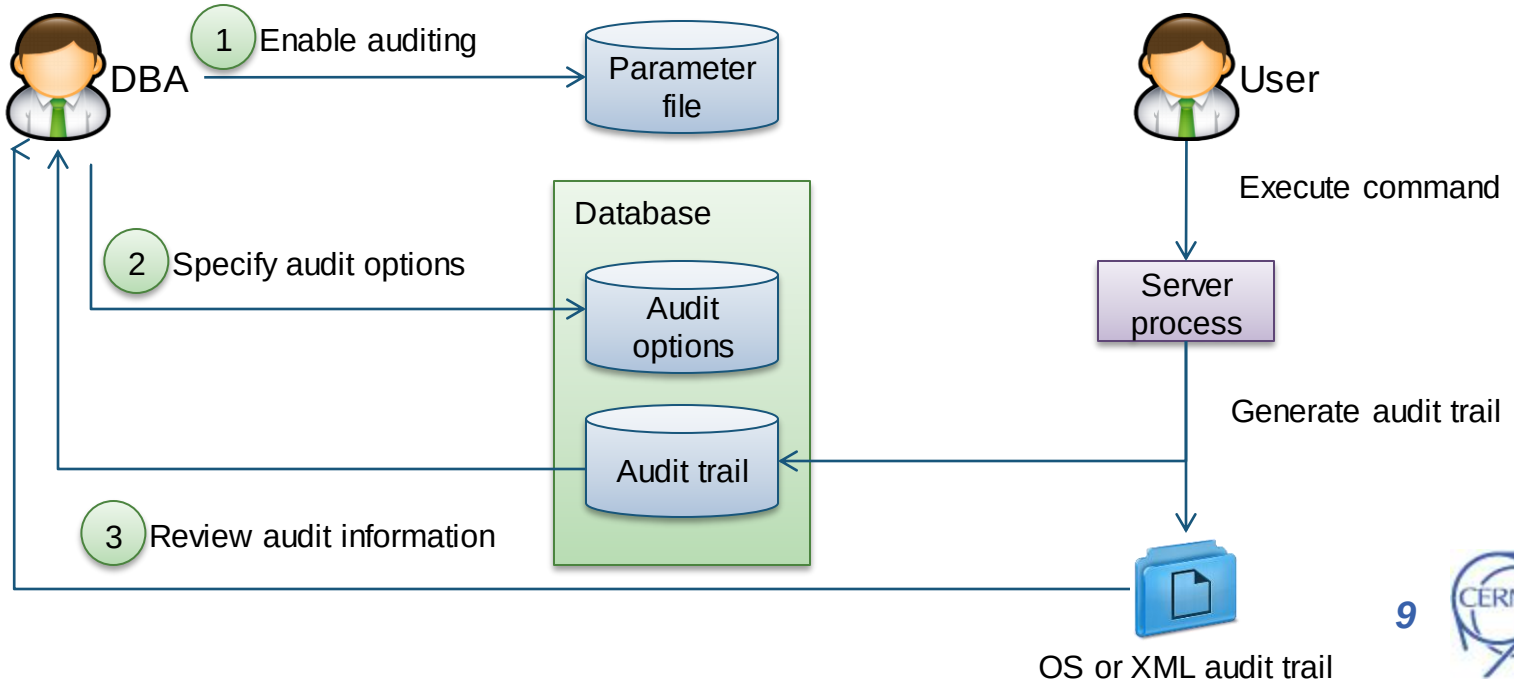
```
CREATE ROLE secure_application_role [IDENTIFIED USING
    <security_procedure_name>];
ALTER USER username
    DEFAULT ROLE (role | NONE | ALL [EXCEPT role1,
    role2, …]);
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

CERN

Auditing

- Capturing and storing information about what is happening
- Identifying unusual database activity
- Several configurations
  - Mandatory auditing
  - Standard database auditing
  - Fine-grained auditing (FGA)
  - Value-based auditing
  - Administrator auditing



CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Auditing: standard auditing

- Set `AUDIT_TRAIL` parameter to storage location
  - `OS`
    - On Windows: event log
    - On UNIX or Linux: `AUDIT_FILE_DEST` parameter
  - `DB` (or `DB, EXTENDED`): `DBA_AUDIT_TRAIL` view, part of the `SYS` schema
  - `XML` (or `XML, EXTENDED`): `AUDIT_FILE_DEST` parameter
    - `V$XML_AUDIT_TRAIL` view contains XML files in directory
- Audit trail must be maintained (might reduce performance)
- Oracle recommends `OS` audit trails
- SQL statement auditing (DDL)
  - `AUDIT TABLE;`
  - `AUDIT TABLE BY user WHENEVER NOT SUCCESSFUL;`
- System-privilege auditing
  - `AUDIT SELECT ANY TABLE, CREATE ANY TRIGGER;`
  - `AUDIT SELECT ANY TABLE BY USER BY SESSION;`
- Object-privilege auditing
  - `AUDIT ALL ON schema.table;`
  - `AUDIT UPDATE,DELETE ON schema.table BY user BY ACCESS;`

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

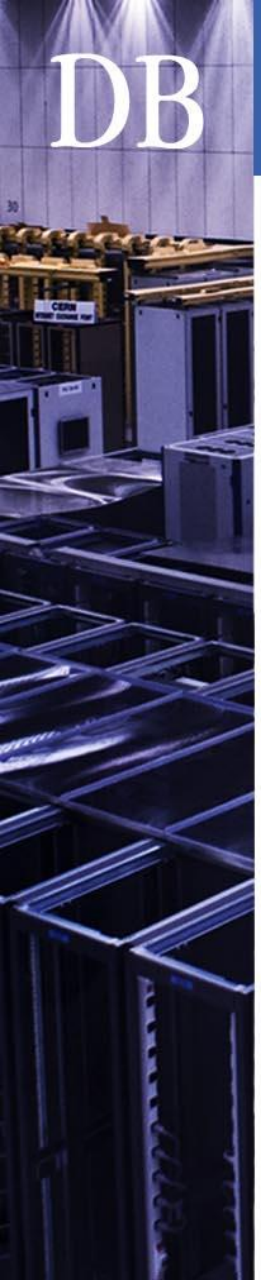
Auditing: default auditing

- Some privileges and statements audited by default

Privileges Audited by Default		
ALTER ANY PROCEDURE	CREATE ANY LIBRARY	GRANT ANY PRIVILEGE
ALTER ANY TABLE	CREATE ANY PROCEDURE	GRANT ANY ROLE
ALTER DATABASE	CREATE ANY TABLE	DROP ANY PROCEDURE
ALTER PROFILE	CREATE EXTERNAL JOB	DROP ANY TABLE
ALTER SYSTEM	CREATE PUBLIC DATABASE LINK	DROP PROFILE
ALTER USER	CREATE SESSION	DROP USER
AUDIT SYSTEM	CREATE USER	EXEMPT ACCESS POLICY
CREATE ANY JOB	GRANT ANY OBJECT PRIVILEGE	

Statements audited by Default
SYSTEM AUDIT BY ACCESS
ROLES BY ACCESS

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



Auditing: FGA

- Fine-Grained Auditing (FGA)
  - Data access monitoring base on content
  - Audits `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `MERGE`
  - Can be set to audit specific columns
  - May execute procedures
  - Administered with `DBMS_FGA` package
- An FGA policy defines the audit criteria and action

```
dbms_fga.add_policy (
     object_schema => 'schema',
     object_name => 'table',
     policy_name => 'audit_example',
     audit_condition => 'id=10',
     audit_column => 'column_name',
     handler_schema => 'secure',
     handler_module => 'log_example',
     enable => TRUE,
     statement_types=> 'SELECT,UPDATE');
```

- Considerations
  - `DELETE` statements are audited regardless of specified columns
  - `MERGE` statements are audited with the corresponding `INSERT`, `UPDATE` and `DELETE` statements

Auditing: value based auditing

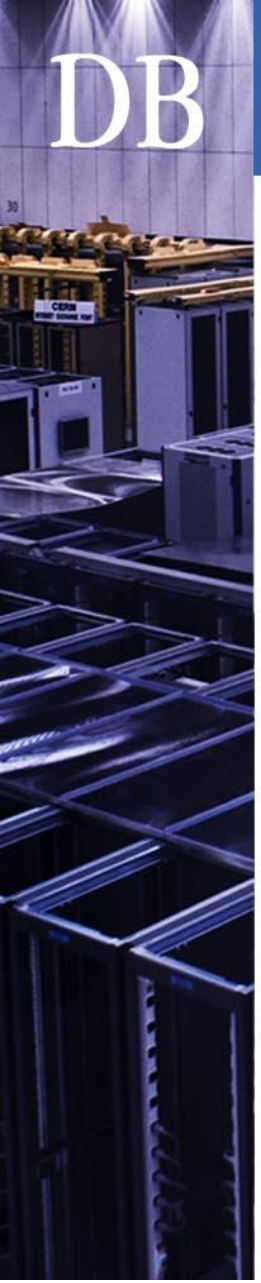
CERN IT Department

- Capture changed values, not just executed actions
- Database triggers are used (degraded performance)
- To be used in special cases
  - Logic behind
  - Standard database auditing is not sufficient
  - Parameters obtained using `SYS_CONTEXT` function
- Example

```
CREATE OR REPLACE TRIGGER system.example_audit
    AFTER UPDATE OF column_name ON schema.table
    REFERENCING NEW AS NEW OLD AS OLD
    FOR EACH ROW
    BEGIN
          IF :old.column_name < :new.column_name
          THEN
              INSERT INTO system.audit_example VALUES
               (sys_context('userenv','os_user'), sysdate,
               sys_context('userenv','ip_address'),
               :new.id || ' changed from
               '||:old.column_name|| ' to '||:new.column_name);
          END IF;
END;
```

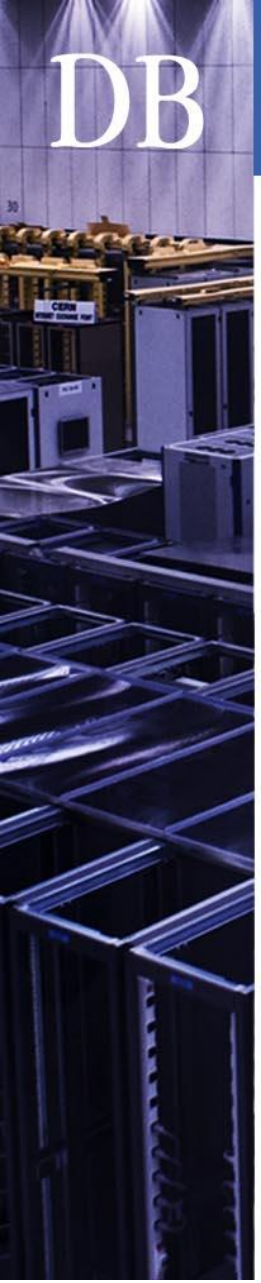
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

CERN



Auditing: administrator auditing

- Users with `SYSDBA` and `SYSOPER` privileges can connect when database is closed
- Connections `AS SYSDBA` and `AS SYSOPER` are always audited
- Other actions can be audited
  - `AUDIT_SYS_OPERATIONS=TRUE` in parameters file
- Audit trail stored outside the database
  - On Windows: event log
  - On UNIX or Linux: `AUDIT_FILE_DEST` parameter



Auditing: securing audit information

CERN IT Department

- Archived audit information must be secured
  - Extract data to location on local disk
  - Encrypt data (digitally signed by the auditing system)
- Audit the audit system
- Oracle Audit Vault
  - Collects and consolidates audit data, beginning with
    - Oracle 9i Database Release 2
    - Microsoft SQL Server 2000 & 2005
    - IBM DB2 Unix, Linux, Windows 8.2 & 9.5
    - Sysbase ASE 12.5 – 15.0
  - Built-in and custom reports
  - Open audit schema to be accessed from
    - Oracle BI Publisher
    - Oracle Application Express
    - Third-party reporting tools
  - Secure and scalable audit warehouse

Database security tutorial

Part II

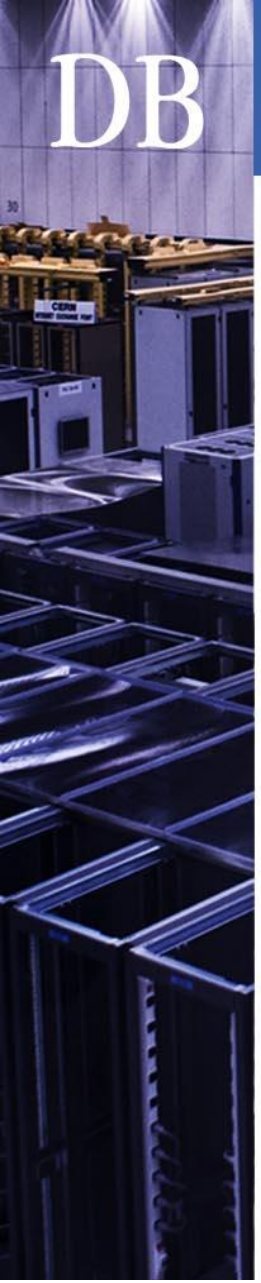
Oracle Tutorials, June 4th 2012

Szymon Skorupinski

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

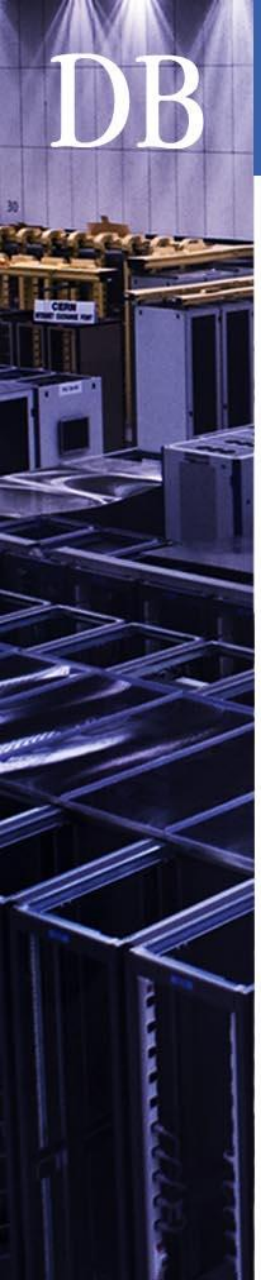
Database SERVICES

CERN



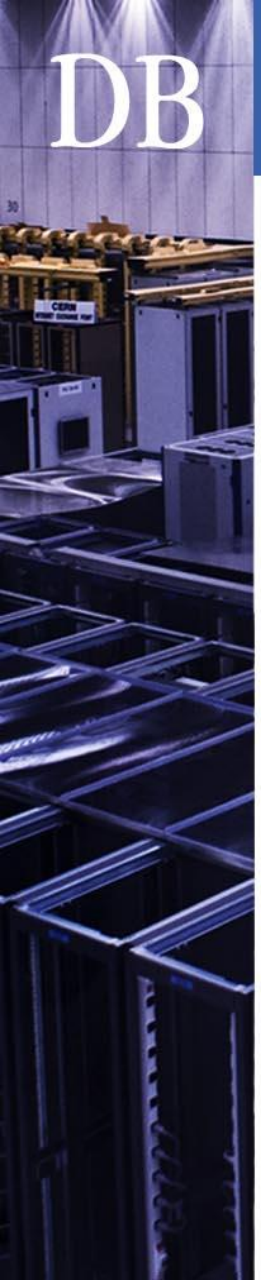
Agenda

- Encryption in Oracle databases
- SQL injection attacks in Oracle
- Questions



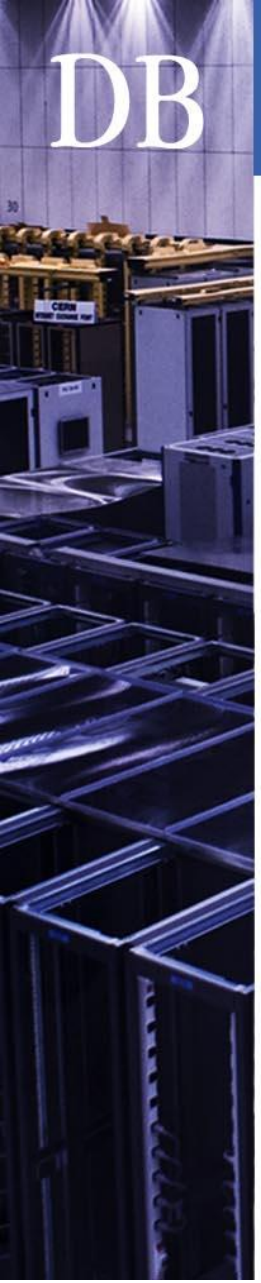
Encryption in Oracle DBs (1/2)

- Way to increase protection for sensitive data
- Oracle Enterprise Edition with Advanced Security option required
- Transparent Data Encryption (TDE)
  - No application changes needed
  - Encryption of data before it's written to storage
  - Decryption of data when it's read from storage
- Two modes supported
  - Tablespace encryption (11g) – hardware acceleration possible
  - Column encryption (10gR2)



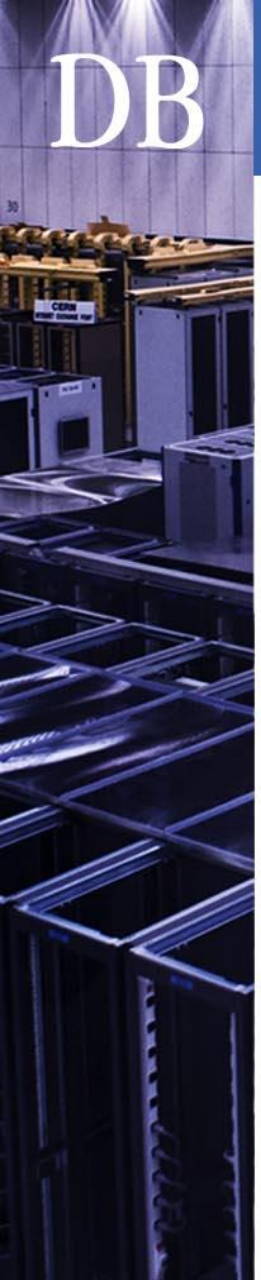
Encryption in Oracle DBs (2/2)

- Additional protection for data in transit
  - Network encryption to protect communication to and from the database
  - Rejecting connections from clients without encryption
- Additional protection for backups
  - TDE encrypted data remains encrypted
  - Entire backup and export dump files encryption possibility



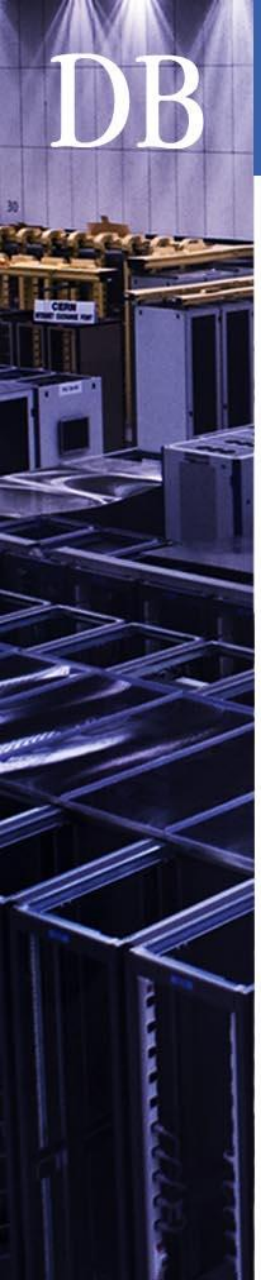
SQL injection defined

- Kind of attack with adding and executing unintended code from untrusted source
  - Manipulate select statements
  - Run DML or even DDL
  - Run stored procedures
- Virtually anything could be done in context of connected user privileges
  - Even more with definer's right procedures
- Caused by
  - Wrong input handling – not only strings!
  - Implicit types conversions – dangerous



SQL injection prevention (1/2)

- Design security into your application from day 1
  - Detection very hard and time consuming in post-development phase
  - Could procedure without any input parameters be injected? Yes...
- Use bind variables!
  - You'll be secure...
  - ...and will get better performance and scalability
- If not...
  - „then you must submit your code for review to at least five people who do not like you - they must be motivated to rip your code apart, critically review it, make fun of it - so they find the bugs" Tom Kyte



SQL injection prevention (2/2)

- If you really have very good technical reasons not to use binds
  - Are you sure?
  - Use DBMS_ASSERT package to sanitize user inputs
  - Are you 100% sure?
- Don't use implicit types conversions...
- ...and don't rely on defaults
  - Application logic unintended change besides SQL injections

SQL injection – be prepared!

Source: niebezpiecznik.pl

SQL injection inputs example (1/7)

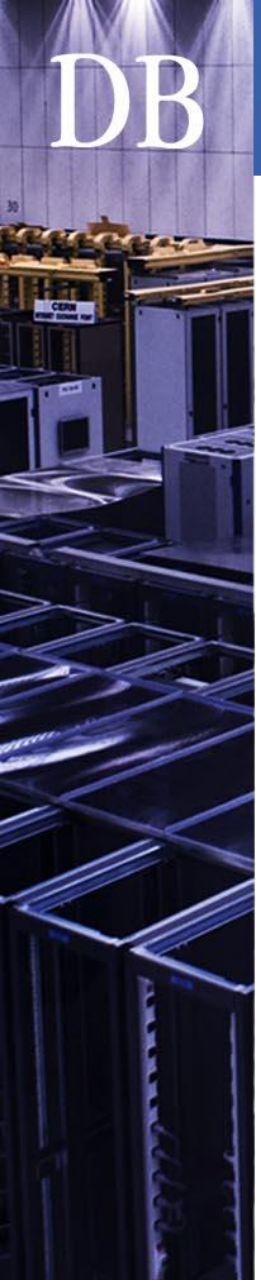

```
SQL> create table users (
       login varchar2(20),
       pass varchar2(20)
     );
Table created.
SQL> insert into users values ('admin','pass');
1 row created.
SQL> commit;
Commit complete.

SQL> select 1 out from users where login = 'admin' and
     pass ='fake';
no rows selected
SQL> select 1 out from users where login = 'admin' and
     pass = 'pass';
            OUT
--------------
             1
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

SQL injection inputs example (2/7)


```
SQL> select 1 out from users where login = '&usr' and
     pass = '&pwd';

Enter value for usr: admin
Enter value for pwd: fake' or 'a'='a

old   1: select 1 out from users where login = '&usr'
         and pass = '&pwd'
new   1: select 1 out from users where login = 'admin'
         and pass = 'fake' or 'a'='a'

            OUT
--------------
             1
```

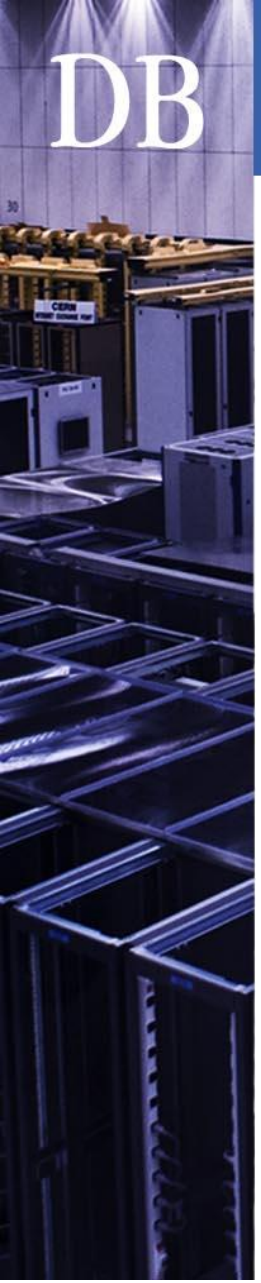
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL injection inputs example (3/7)


```
SQL> variable usr varchar2(20);
SQL> variable pwd varchar2(20);

SQL> exec :usr := 'admin';

PL/SQL procedure successfully completed.

SQL> exec :pwd := 'fake'' or ''a'' = ''a';

PL/SQL procedure successfully completed.

SQL> select 1 out from users where login = :usr and pass
     = :pwd;

no rows selected
```

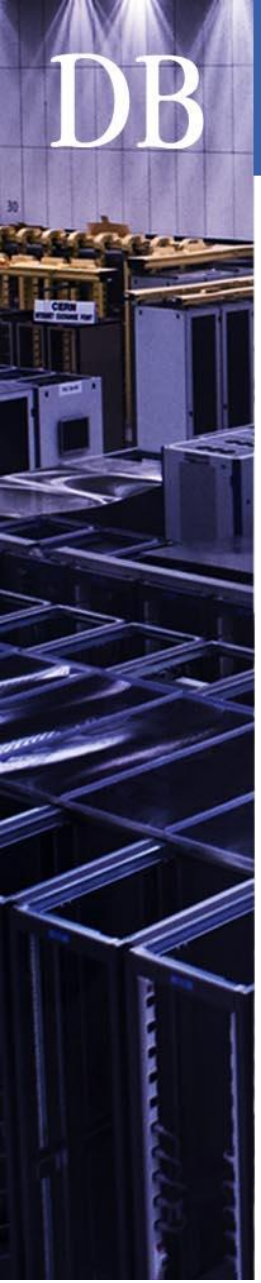
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL injection inputs example (4/7)


```
SQL> create or replace procedure add_user (p_login
     varchar2, p_pass varchar2) as
       l_cmd varchar2(1000);
     begin
       l_cmd := '
         begin
           insert into users values (''' || p_login ||
           ''','''' || p_pass || ''');
           commit;
         end;';
       dbms_output.put_line(l_cmd);
       execute immediate l_cmd;
     end;
     /

Procedure created.
```

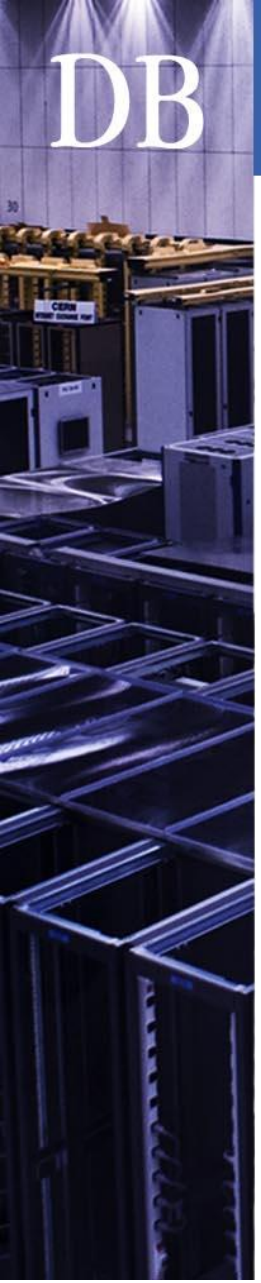
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL injection inputs example (5/7)


```
SQL> set serveroutput on
SQL> select * from users;
LOGIN                PASS
-------------------- --------------------
admin                pass

SQL> exec add_user('NewLogin','NewPass');
  begin
    insert into users values ('NewLogin','NewPass');
    commit;
  end;
PL/SQL procedure successfully completed.

SQL> select * from users;
LOGIN                PASS
-------------------- --------------------
admin                pass
NewLogin             NewPass
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL injection inputs example (6/7)


```
SQL> exec add_user('NewerLogin','NewerPass''); insert
     into users values (''FakeUser'', ''FakePass'');--');
begin
  insert into users values ('NewerLogin', 'NewerPass');
  insert into users values ('FakeUser', 'FakePass');--');
  commit;
end;
PL/SQL procedure successfully completed.

SQL> select * from users;
LOGIN                PASS
-------------------- --------------------
NewerLogin           NewerPass
admin                pass
NewLogin             NewPass
FakeUser             FakePass
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

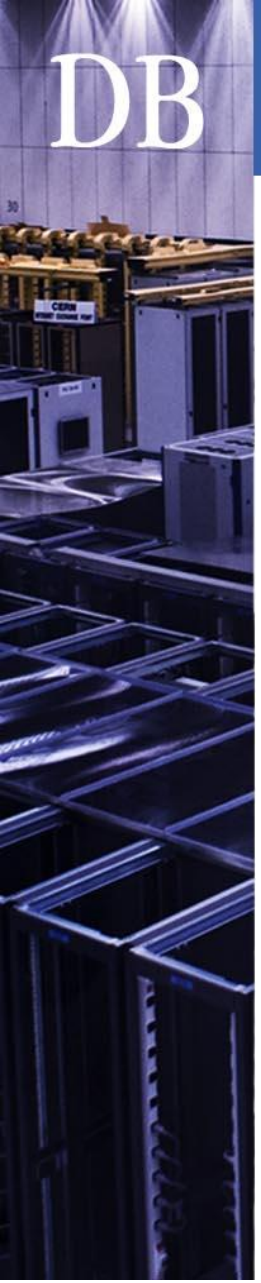
SQL injection inputs example (7/7)


```
SQL> exec add_user('NewestLogin','NewestPass'');
     execute immediate ''drop table users'';--');
begin
  insert into users values ('NewestLogin','NewestPass');
  execute immediate 'drop table users';--');
  commit;
end;

PL/SQL procedure successfully completed.

SQL> select * from users;
select * from users
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

SQL injection no inputs example (1/6)

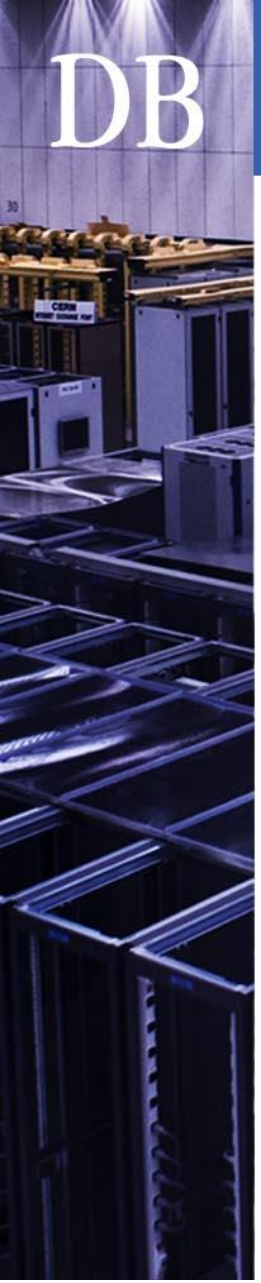

```
SQL> create table users (
       login  varchar2(30),
       pass   varchar2(30),
       expire timestamp
     );

Table created.

SQL> alter session set nls_timestamp_format = 'DD-MM-
     YYYY HH24:MI:SS';

Session altered.
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL injection no inputs example (2/6)


```
SQL> insert into users values ('UserExpired',
     'pass1234', systimestamp - 1);
1 row created.

SQL> insert into users values ('UserNotExpired',
     '4567pass', systimestamp + 1);
1 row created.

SQL> commit;
Commit complete.

SQL> select * from users;
LOGIN          PASS           EXPIRE
-------------- -------------- --------------------
UserExpired    pass1234       25-05-2012 22:07:35
UserNotExpired 4567pass       27-05-2012 22:07:35
```



SQL injection no inputs example (3/6)


```
SQL> create or replace procedure list_expired_users is
       l_query       varchar2(300);
       l_query_bind  varchar2(300);
       l_time        timestamp;
       l_cur         sys_refcursor;
       l_login       varchar2(30);
     begin
       l_time := systimestamp;
       l_query := 'select login from users where expire
       <= ''' || l_time || '''';
       l_query_bind := 'select login from users where
       expire <= :b_var';

       dbms_output.put_line('Concatenated query with
       implicit conversions: ' || l_query);
       open l_cur for l_query;
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

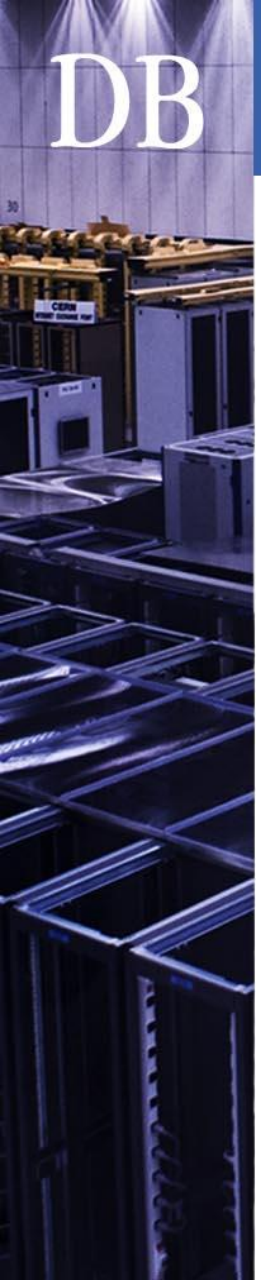
SQL injection no inputs example (4/6)


```
  loop
    fetch l_cur into l_login;
    exit when l_cur%notfound;
    dbms_output.put_line(l_login);
  end loop;
  close l_cur;

  dbms_output.put_line('Bind variable query: ' ||
  l_query_bind);
  open l_cur for l_query_bind using l_time;
  loop
    fetch l_cur into l_login;
    exit when l_cur%notfound;
    dbms_output.put_line(l_login);
  end loop;
  close l_cur;
end;
/
```

SQL injection no inputs example (5/6)


```
SQL> show parameter nls_timestamp_format

NAME                           TYPE        VALUE
------------------------------ ----------- ------------------------
nls_timestamp_format           string      DD-MM-YYYY HH24:MI:SS

SQL> set serverout on
SQL> exec list_expired_users;

Concatenated query with implicit conversions: select
login from users where expire <= '26-05-2012 22:14:24'
UserExpired

Bind variable query: select login from users where
expire <= :b_var
UserExpired

PL/SQL procedure successfully completed.
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



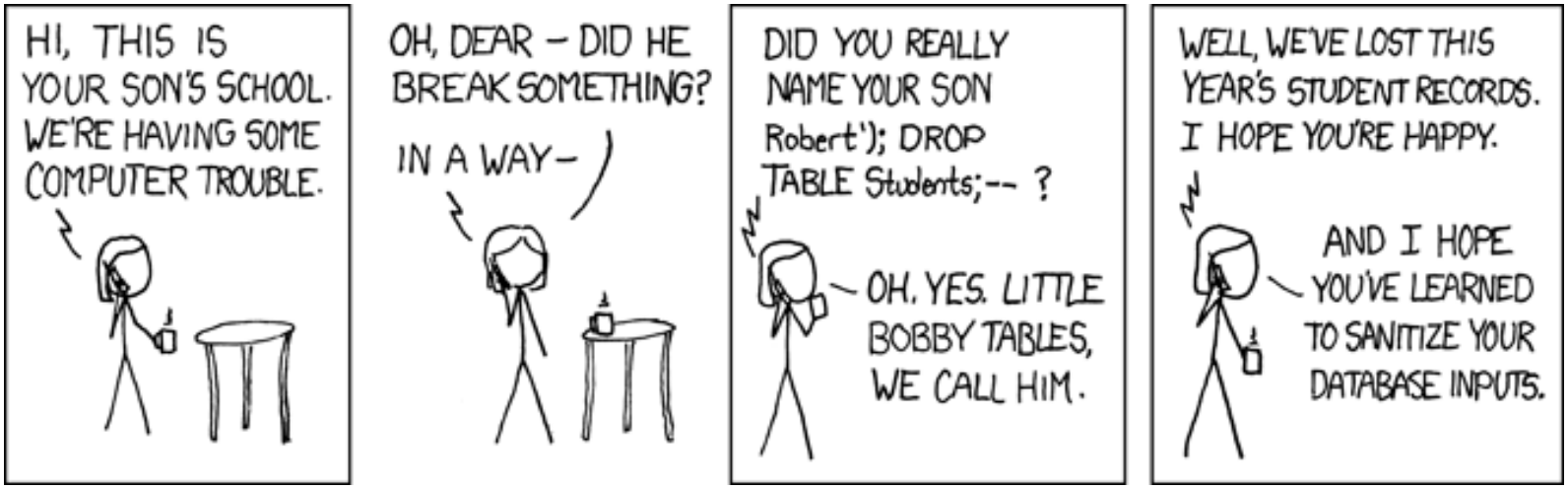
SQL injection no inputs example (6/6)


```
SQL> alter session set nls_timestamp_format = '"'' union
     select login || '' '' || pass from users--"';
Session altered.

SQL> exec list_expired_users;
Concatenated query with implicit conversions:
select login from users where expire <= '' union select
login || ' ' || pass from users--'
UserExpired pass1234
UserNotExpired 4567pass

Bind variable query:
select login from users where expire <= :b_var
UserExpired

PL/SQL procedure successfully completed.
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Questions?

Thank you!



Source: xkcd.com

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it