SQL Injection Prevention

May 3rd 2012
About Me

• Tom Webb
  – webbtc@mailbox.sc.edu
  – 803-777-1701

• 12 Years in IT

• Over 6 years at USC

• Irhowto.wordpress.com

• computer-forensics.sans.org/blog
Who should Attend this Talk

• DBA’s
• Web Programmers
What We are going to Cover

• Brief Discussion of what is SQL Injection
• SQL Application Side
• Language Side (PHP, ASP)
• OS Side
What we are NOT going to cover

• Hacking techniques using SQL Injection
• Tons of presentations on how to do this
What is SQL Injection?

SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution

Where SQL Injection Takes place?

• Any place you take input from a user and its passed to the database.
Simple Select Statement

Tbl_users

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
<th>Address</th>
<th>passwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td><a href="mailto:bob@aol.com">bob@aol.com</a></td>
<td>77101</td>
<td>12 main st</td>
<td>1234</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td><a href="mailto:jim@aol.com">jim@aol.com</a></td>
<td>7-0001</td>
<td>32 sun st</td>
<td>pass</td>
</tr>
</tbody>
</table>

- select email from tbl_user where name = 'bob';

bob@aol.com
SQL injection

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
<th>Address</th>
<th>passwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td><a href="mailto:bob@aol.com">bob@aol.com</a></td>
<td>77101</td>
<td>12 main st</td>
<td>1234</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td><a href="mailto:jim@aol.com">jim@aol.com</a></td>
<td>7-0001</td>
<td>32 sun st</td>
<td>pass</td>
</tr>
</tbody>
</table>

select email from tbl_user where name = 'bob' and union select name, passwd from tbl_user;--'

Bob,1234
Jim,pass
Escalating the Vulnerability on any Platform

• Get data from the any Table/Database the user rights allowed

• Write Files to the Server. (Backdoors)
  – Union select ‘nc -l -p 3333 -e /bin/bash’,, into output file /etc/cron.hourly/backup.sh

• Read file on the server
  – UNION SELECT LOAD_FILE('/etc/passwd'),

• Execute commands on the server
Prevention

• SQL Application Side
  – SQL User Database Rights
    • Limit right to just select for specific users
  – Stored Procedures
• Language Side (PHP, ASP)
• OS Side
  – Limit rights of the service/user running
  – App Armor (Ubuntu) or SELinux (CentOS)
Prevention (2)

• DO NOT STORE SENSITIVE DATA!
• If you need to, don’t store on system accessible from the Internet.
PROTECTION WITHIN SQL
Prepared Statement (Parameterized Queries)

- Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later.

- This coding style allows the database to distinguish between code and data, regardless of what user input is supplied.

(http://owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
Prepared Statement (2)

• MySQL
  – 'Select * from table where name is ?'
• Oracle PL/SQL
  – 'SELECT * FROM table where name=:1'
• MSSQL
  – "SELECT * FROM table WHERE name = ?"
MySQL Prepared Statement

prepare id from
'select Id from table where name= ?' ;
set @mynname :='mycomputer' ;
execute id @mynname;
DEALLOCATE PREPARE id ;
MySQL Procedure

DROP PROCEDURE IF EXISTS `table`.`get_id`;
DELIMITER $$
CREATE PROCEDURE `IR`.`get_id`(IN input VARCHAR(20), OUT id VARCHAR(5))
READS SQL DATA
BEGIN
SET @input=input;
prepare id from
'\select Id from table where name= ?' ;
execute id USING @input;
DEALLOCATE PREPARE id;
END$$
MySQL Procedure (2)

• `mysql> call get_id('mycomputer', @out);`

+--------+
| @out    |
+--------+
|     1   |
+--------+
MYSQL Stored Procedure Troubleshooting

• Print output using select to troubleshoot
  – Select @variable;

• Set GLOBAL general_log= ‘ON’;
• Set GLOBAL log_output = ‘table’;
• Select * from mysql.general_log limit 50;
• Don’t forget to turn this off!
Oracle PL/SQL

DECLARE
    username varchar2(32);
    password varchar2(32);
    result integer;
BEGIN
    Execute immediate 'SELECT count(*) FROM users where username=:1 and password=:2' into result using username,password;
END;

(Clarke, 2009, Ch. 8)
CREATE PROCEDURE SP_ProductSearch @prodname varchar(400) = NULL AS

DECLARE @sql nvarchar(4000)

SELECT @sql = 'SELET ProductID, ProductName, Category, Price' + ' FROM Product Where '

IF @prodname IS NOT NULL
    SELECT @sql = @sql + ' ProductName LIKE @prodname'

EXEC sp_executesql @sql, N'@prodname varchar(400)', @prodname
(palizine.plynt.com)
Implementation Strategy

• Highest Protection level.
• Best for client side code.
• Most likely this will be implemented for new applications.

• Pro
  – Logic built into the database. Client side independent.
PREPARED STATEMENTS WITH OTHER LANGUAGES
PHP

• $con = new mysqli("localhost", "username", "password", "db");
• $sql = "SELECT * FROM users WHERE username=? AND password=?";
• $cmd = $con->prepare($sql);
• // Add parameters to SQL query
• $cmd->bind_param("ss", $username, $password); // bind parameters as strings
• $cmd->execute();
  – (Clarke, 2009, Ch. 8)
PHP PDO Package

- $sql = “SELECT * FROM users WHERE username=:username
  AND” + “password=:password”;
- $stmt = $dbh->prepare($sql);
- // bind values and data types
- $stmt->bindParam(':username', $username, PDO::PARAM_STR, 12);
- $stmt->bindParam(':password', $password, PDO::PARAM_STR, 12);
- $stmt->execute();

(Clarke, 2009, Ch. 8)
.net

String query =
"SELECT account_balance FROM user_data WHERE user_name = ?";
try {
    OleDbCommand command = new OleDbCommand(query, connection);
    command.Parameters.Add(new OleDbParameter("customerName", CustomerName Name.Text));
    OleDbDataReader reader = command.ExecuteReader();
} catch (OleDbException se) {
    // error handling
}

(https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
Implementation Strategy

• Highest Protection level.

• Most likely this will be implemented by replacing the dynamic sql queries with prepared statements.

• Pro
  – Easier to implement then SQL server.
ESCAPING USER INPUT
Oracle Escape

Codec ORACLE_CODEC = new OracleCodec();
String query = "SELECT user_id FROM user_data WHERE user_name = "" +
    ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("userID")) + "" and user_password = "" +
    ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("pwd")) +"";

(https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
<td>
  <input type="text" runat=server id="txtName"/>
  <asp:RegularExpressionValidator runat=server
    ControlToValidate="txtName"
    ErrorMessage="ID must be 6-10 letters."
    ValidationExpression="[a-zA-Z]{6,10}" />
</td>

(http://msdn.microsoft.com/en-us/library/ms972961.aspx)
// Strip slashes
if (get_magic_quotes_gpc())
{
    $value = stripslashes($value);
}
if (!is_numeric($value))
{
    $value = "" . mysql_real_escape_string($value) . "" ;
}
return $value;

• // Make a safe SQL
$user = check_input($_POST['user']);
$pwd = check_input($_POST['pwd']);
$sql = "SELECT * FROM users WHERE user=$user AND password=$pwd";
mysql_query($sql);
mysql_close($con);
PHP Validation

• $username = $_POST['username'];
• if (!preg_match("/^[a-zA-Z]{8,12}$/D", $username)) {
  • // handle failed validation
• }
• (Clarke, 2009, Ch. 8)
Implementation Strategy

• Medium Protection level.
  – Can be bypassed, best to whitelist input.

• Should be included along with prepared statements.

• Pro
  – Can be added to legacy code quickly (Band-Aid).
APPLICATION HARDENING
Disable Error Messages

• Asp.net web.config
  – <configuration>
  – <system.web>
  – <customErrors mode="RemoteOnly">
  – <compilation debug="false">

• IIS
  – Double click on website from IIS snap-in select custom error tab
Disable Error Messages (2)

• PHP
  – Edit php.ini
    • set display_errors=off
    • html_errors = Off

• Apache
  – Httpd.conf
  – ErrorDocument 500 "Sorry, our script crashed. Oh dear"

(http://httpd.apache.org/docs/2.0/custom-error.html)
Limit DB Users Access

• Only allow user access to what it needs.
  – If you only do select on one table give just select rights.

• Create a unique user per web application/database
Limit DB Users Access(2)

• MySQL
  – Grant select (ID,name) on DB.Table to ‘user’@’%’;
  – Grant execute ON Procedure `table`.`procedure` to ‘user’@’%’;

• MSSQL
  – GRANT SELECT ON OBJECT::DB.Table TO user;
  – GRANT EXECUTE ON OBJECT::Db.Table TO user;
Limit DB Users Access(2)

- ORACLE
  - grant select, on table to user;
  - grant execute on Procedure to user;
Blocking External Access

• Apache.config
  <Directory /var/www/phpmyadmin/>
  Order deny,allow
  Deny from all
  Allow from 129.252.41.0/24
  Allow from 127.0.0.1
  </Directory>

• IIS
  – http://support.microsoft.com/kb/324066
Hardening Guide

• [https://benchmarks.cisecurity.org/en-us/?route=downloads.multiform](https://benchmarks.cisecurity.org/en-us/?route=downloads.multiform)

• Great Information, but could break your application.

• BACKUP YOUR CONFIGURATIONS BEFORE USING THEM!!!!
Implementation Strategy

• All these configuration setting should be applied.

• Pro
  – Easy to implement without changing code.
PATCHING!!!!!
What to Patch

- Services (Oracle, MySQL, Apache, IIS, ect...)
- Content Management Systems (Joomla!)
- Languages (PHP, Perl....)
- Web Applications (PHPmyadmin, webmin, Calendar apps, Forums, Wordpress)
- Subscribe to RSS, email lists, forums that have patch notifications.
ADDITIONAL PROTECTION
Rule Based Application Protections

• Mod_Security
  – Apache module that has rules to block attacks

• URL SCAN
  – IIS module that has rules to block attacks

• GreenSQL
  – MySQL and Postgres SQL Proxy
Implementation Strategy

• Defense in Depth
• Band-Aid approach if you are not allowed to change anything on a server, but keep it running.
References


• [http://www.w3schools.com/php/func_mysql_real_escape_string.asp](http://www.w3schools.com/php/func_mysql_real_escape_string.asp)