SQL Injection

Slides thanks to Prof. Shmatikov at UT Austin
Dynamic Web Application

GET / HTTP/1.0

HTTP/1.1 200 OK

index.php

Database server
PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  
  ```html
  <input value='<?php echo $myvalue; ?>'>
  ```

- Can embed variables in double-quote strings
  ```
  $user = "world"; echo "Hello $user!";
  ```
  or
  ```
  $user = "world"; echo "Hello" . $user . "!";
  ```

- Form data in global arrays `$_GET`, `$_POST`, ...
- Widely used database query language
- Fetch a set of records
  
  \[
  \text{SELECT} \ * \ \text{FROM} \ \text{Person} \ \text{WHERE} \ \text{Username}=\text{`Vitaly`} 
  \]
- Add data to the table
  
  \[
  \text{INSERT INTO} \ \text{Key} \ (\text{Username}, \ \text{Key}) \ \text{VALUES} \ (\text{`Vitaly’}, \ 3611BBFF) 
  \]
- Modify data
  
  \[
  \text{UPDATE} \ \text{Keys} \ \text{SET} \ \text{Key}=\text{FA33452D} \ \text{WHERE} \ \text{PersonID}=5 
  \]
- Query syntax (mostly) independent of vendor
Sample PHP Code

Sample PHP

```php
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " . "WHERE Username='\$selecteduser'";
$rs = $db->executeQuery($sql);
```

What if `user` is a malicious string that changes the meaning of the query?
**SQL Injection: Basic Idea**

1. **Attacker** post malicious form.
2. **Victim SQL DB** unintended query.
3. **Victim server** receive valuable data.

- This is an **input validation vulnerability**
  - Unsanitized user input in SQL query to back-end database changes the meaning of query.
- Specific case of more general command injection.
Typical Login Prompt
User Input Becomes Part of Query

Web browser (Client) → Enter Username & Password → Web server → SELECT passwd FROM USERS WHERE uname IS 'user' → DB
Normal Login

Web browser (Client) → Enter Username & Password → Web server → SELECT passwd FROM USERS WHERE uname IS 'smith' → DB
Malicious User Input

![User Login - Microsoft Internet Explorer](image)

```
User Name: '; DROP TABLE USERS; --
Password: ••••••
```
SQL Injection Attack

Web browser (Client) → Web server → DB

Enter Username & Password

SELECT passwd FROM USERS WHERE uname IS ''; DROP TABLE USERS; -- '

Eliminates all user accounts
Exploits of a Mom

http://xkcd.com/327/

1. Hi, this is your son's school. We're having some computer trouble.

2. Oh, dear — did he break something? In a way—

3. Did you really name your son Robert?; Drop Table Students; --?

4. Well, we've lost this year's student records. I hope you're happy.

5. And I hope you've learned to sanitize your database inputs.
Authentication with Back-End DB

- set UserFound=execute("SELECT * FROM UserTable WHERE username='" & form("user") & "' AND password='" & form("pwd") & "' ");

  - User supplies username and password, this SQL query checks if user/password combination is in the database

- If not UserFound.EOF Authentication correct
- else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database
Using SQL Injection to Steal Data

- User gives username ‘ OR 1=1 --
- Web server executes query
  
  ```sql
  set UserFound=execute(
      SELECT * FROM UserTable WHERE
      username=' OR 1=1 -- ... );
  ```

  - Now all records match the query
  - This returns the entire database!
Another SQL Injection Example

To authenticate logins, server runs this SQL command against the user database:

```sql
SELECT * WHERE user='name' AND pwd='passwd'
```

User enters ‘ OR WHERE pwd LIKE ‘%’ as both name and passwd

Server executes

```sql
SELECT * WHERE user=' OR WHERE pwd LIKE '%%'
AND pwd=' ' OR WHERE pwd LIKE '%%'
```

Logs in with the credentials of the first person in the database (typically, administrator!)
It Gets Better

- User gives username

  \texttt{exec cmdshell \textquote{net user badguy badpwd} / ADD --}

- Web server executes query

  \texttt{set UserFound=execute(}
  \texttt{SELECT * FROM UserTable WHERE}
  \texttt{username=\textquote{exec \ldots -- \ldots });}

- Creates an account for badguy on DB server
Pull Data From Other Databases

- User gives username
  
  `AND 1=0
  UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards`

- Results of two queries are combined

- Empty table from the first query is displayed together with the entire contents of the credit card database
More SQL Injection Attacks

- Create new users
  
  `'; INSERT INTO USERS ('uname','passwd','salt') VALUES ('hacker','38a74f', 3234);`

- Reset password
  
  `'; UPDATE USERS SET email=hcker@root.org WHERE email=victim@yahoo.com`
Uninitialized Inputs

/* php-files/lostpassword.php */
for ($i=0; $i<=7; $i++)
    $new_pass .= chr(rand(97,122))
...
$result = dbquery("UPDATE ".$db_prefix."users
    SET user_password=md5("$new_pass")
    WHERE user_id='".$data['user_id']."');

In normal execution, this becomes
UPDATE users SET user_password=md5('?????????')
WHERE user_id='userid'

Creates a password with 8 random characters, assuming $new_pass is set to NULL
SQL query setting password in the DB
Exploit

User appends this to the URL:
&new_pass=badPwd%27%29%2c
user_level=%27103%27%2cuser_aim=%28%27

SQL query becomes
UPDATE users SET user_password=md5('badPwd'),
user_level='103', user_aim=('?????????
WHERE user_id='userid'

This sets $new_pass to badPwd'), user_level='103', user_aim=('?????????

User’s password is set to ‘badPwd’

... with superuser privileges
Second-Order SQL Injection

- Second-order SQL injection: data stored in database is later used to conduct SQL injection

- For example, user manages to set uname to admin' --
  - This vulnerability could exist if string escaping is applied inconsistently (e.g., strings not escaped)
  - UPDATE USERS SET passwd='cracked'
    WHERE uname='admin' --'
  
- Solution: treat all parameters as dangerous
SQL Injection in the Real World (1)

- Oklahoma Department of Corrections divulges thousands of social security numbers (2008)
  - Sexual and Violent Offender Registry for Oklahoma
  - Data repository lists both offenders and employees

- “Anyone with a web browser and the knowledge from Chapter One of SQL For Dummies could have easily accessed – and possibly, changed – any data within the DOC's databases"

http://www.ireport.com/docs/DOC-11831
Ohio State University has the largest enrolment of students in the United States; it also seems to be vying to get the largest number of entries, so far eight, in the Privacy Rights Clearinghouse breach database. One of the more recent attacks that took place on the 31st of March 2007 involved a SQL injection attack originating from China against a server in the Office of Research. The hacker was able to access 14,000 records of current and former staff members.
CardSystems Attack (June 2005)

- CardSystems was a major credit card processing company
- Put out of business by a SQL injection attack
  - Credit card numbers stored unencrypted
  - Data on 263,000 accounts stolen
  - 43 million identities exposed
Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the United Nations and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in Microsoft Windows to install malicious software on visitors’ machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft’s Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn’t aware of anyone trying to exploit that particular weakness.

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.
Main Steps in April 2008 Attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection to modify the pages to include a link to a Chinese site nihaorr1.com
  - Do not visit that site – it serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger
- Attack used automatic tool; can be configured to inject whatever you like into vulnerable sites
- There is some evidence that hackers may get paid for each victim’s visit to nihaorr1.com
DECLARE @T varchar(255),@C varchar(255)
DECLARE Table_Cursor  CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b where
a.id=b.id and a.xtype='u' and
(b.xtype=99 or b.xtype=35 or b.xtype=231 or b.xtype=167)
OPEN Table_Cursor
FETCH NEXT FROM  Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0) BEGIN
  exec('update ['+@T+'] set ['+@C+']=rtrim(convert(varchar,['+@C+']))
  +" "')
FETCH NEXT FROM  Table_Cursor INTO @T,@C
END CLOSE Table_Cursor
DEALLOCATE Table_Cursor;
DECLARE%20@S%20NVARCHAR(4000);SET%20@S=CAST(
%20AS%20NVARCHAR(4000));EXEC(@S);--
Preventing SQL Injection

Input validation

- Filter
  - Apostrophes, semicolons, percent symbols, hyphens, underscores, ...
  - Any character that has special meanings
- Check the data type (e.g., make sure it’s an integer)

Whitelisting

- Blacklisting “bad” characters doesn’t work
  - Forget to filter out some characters
  - Could prevent valid input (e.g., last name O’Brien)
- Allow only well-defined set of safe values
  - Set implicitly defined through regular expressions
Escaping Quotes

➤ For valid string inputs use escape characters to prevent the quote becoming part of the query

- Example: escape(o’connor) = o”connor
- Convert ‘ into \
- Only works for string inputs
- Different databases have different rules for escaping
Prepared Statements

- Metacharacters such as ' in queries provide distinction between data and control.
- In most injection attacks, data are interpreted as control – this changes the semantics of a query or a command.
- Bind variables: ? placeholders guaranteed to be data (not control).
- Prepared statements allow creation of static queries with bind variables. This preserves the structure of intended query.
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
        + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
ResultSet res = ps.executeQuery();

- Query parsed without parameters
- Bind variables are typed (int, string, ...)

http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
Mitigating Impact of Attack

- Prevent leakage of database schema and other information
- Limit privileges (defense in depth)
- Encrypt sensitive data stored in database
- Harden DB server and host OS
- Apply input validation