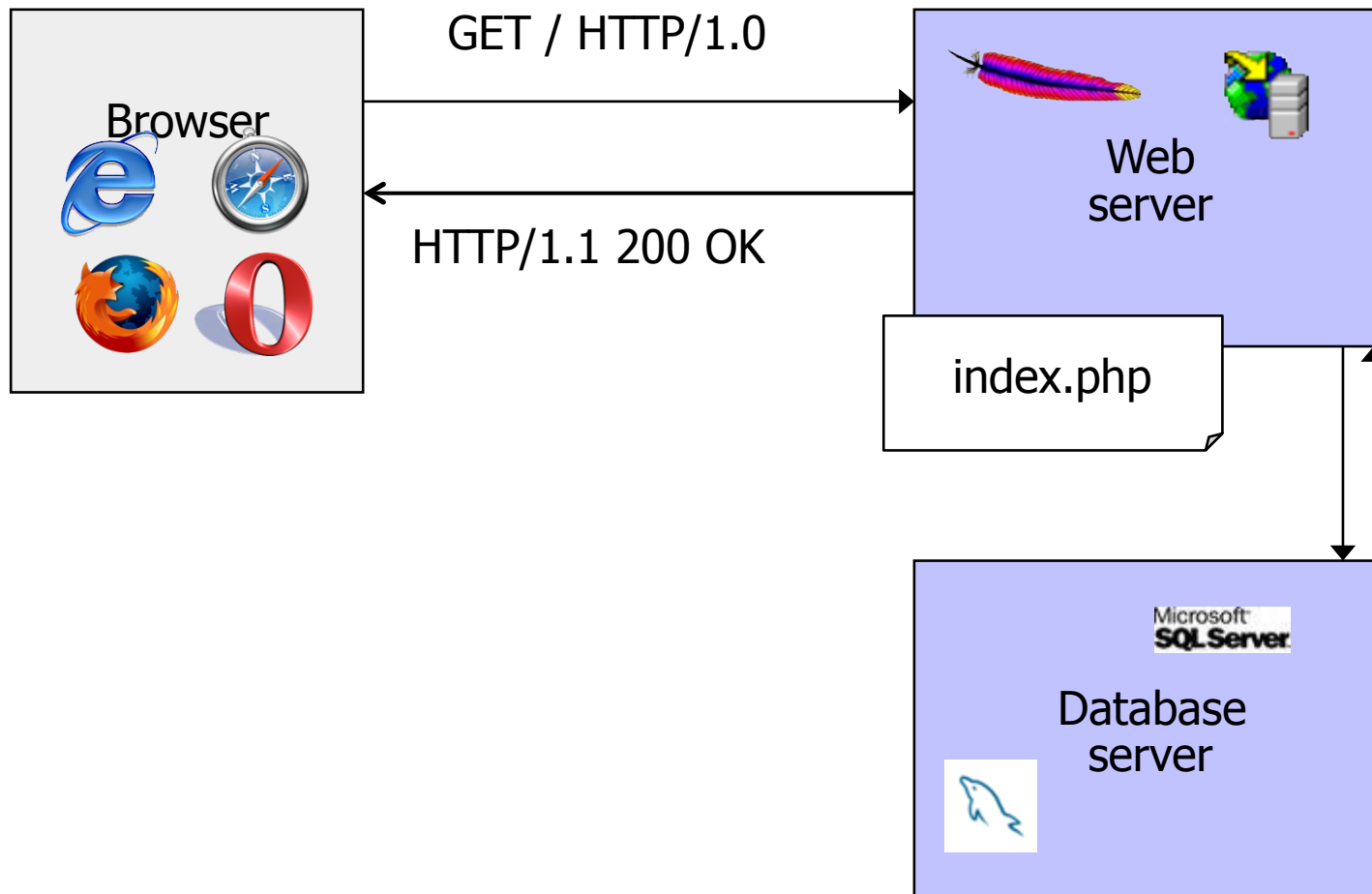


# SQL Injection

Slides thanks to Prof. Shmatikov at UT Austin

# Dynamic Web Application



# PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code

```
<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

```
SELECT * FROM Person WHERE Username='Vitaly'
```

➤ Add data to the table

```
INSERT INTO Key (Username, Key) VALUES ('Vitaly', 3611BBFF)
```

➤ Modify data

```
UPDATE Keys SET Key=FA33452D WHERE PersonID=5
```

➤ Query syntax (mostly) independent of vendor

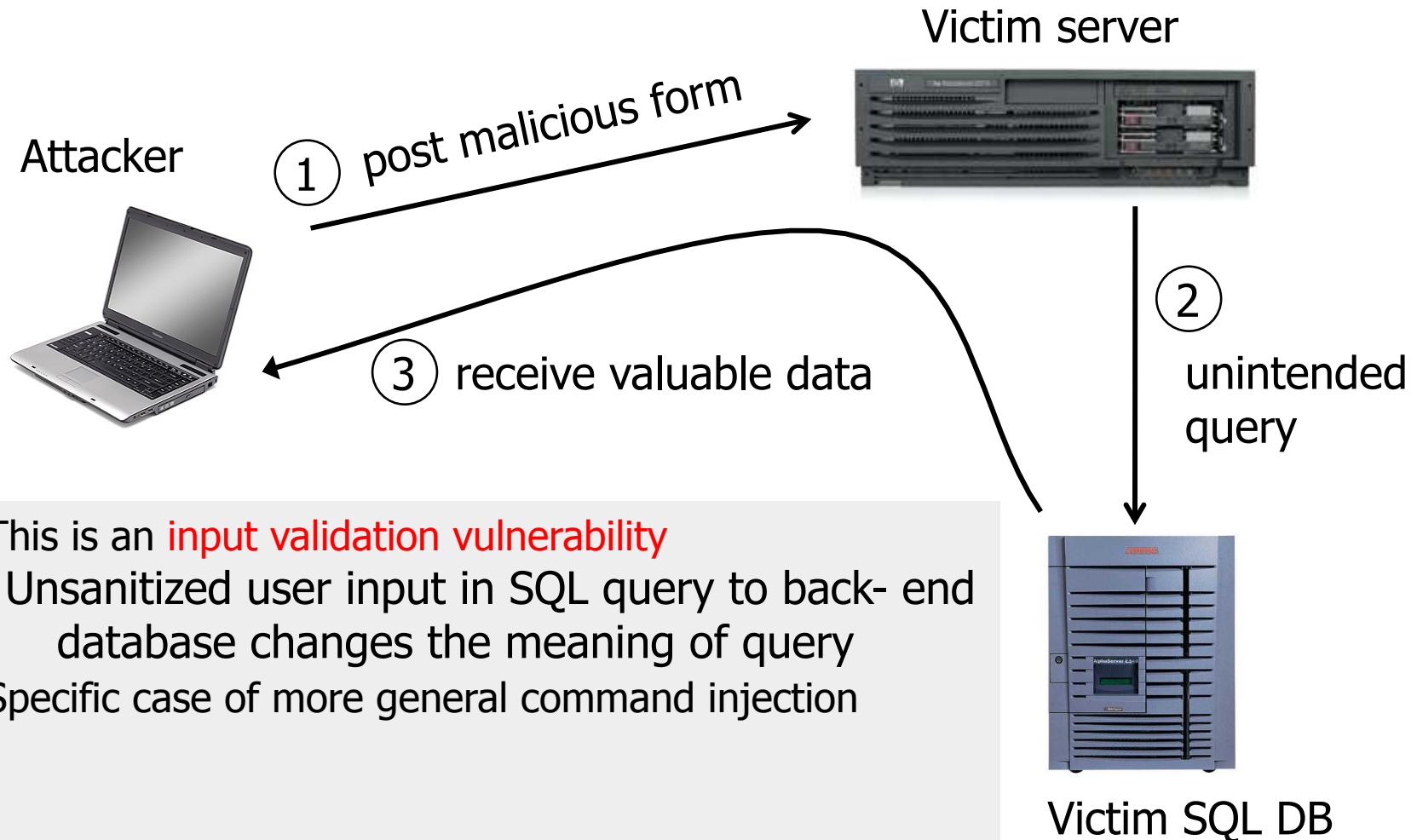
# Sample PHP Code

## ➤ Sample 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



# Typical Login Prompt

User Login - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Reload Home Search

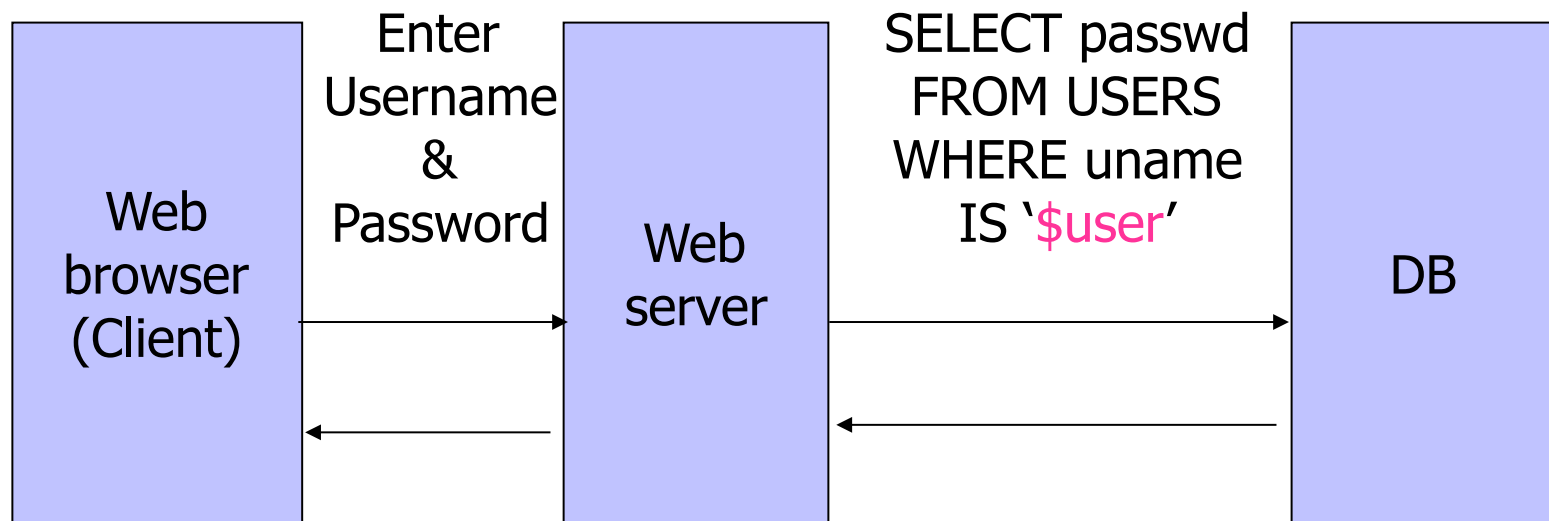
Enter User Name: smith

Enter Password: ●●●●●●●

Login

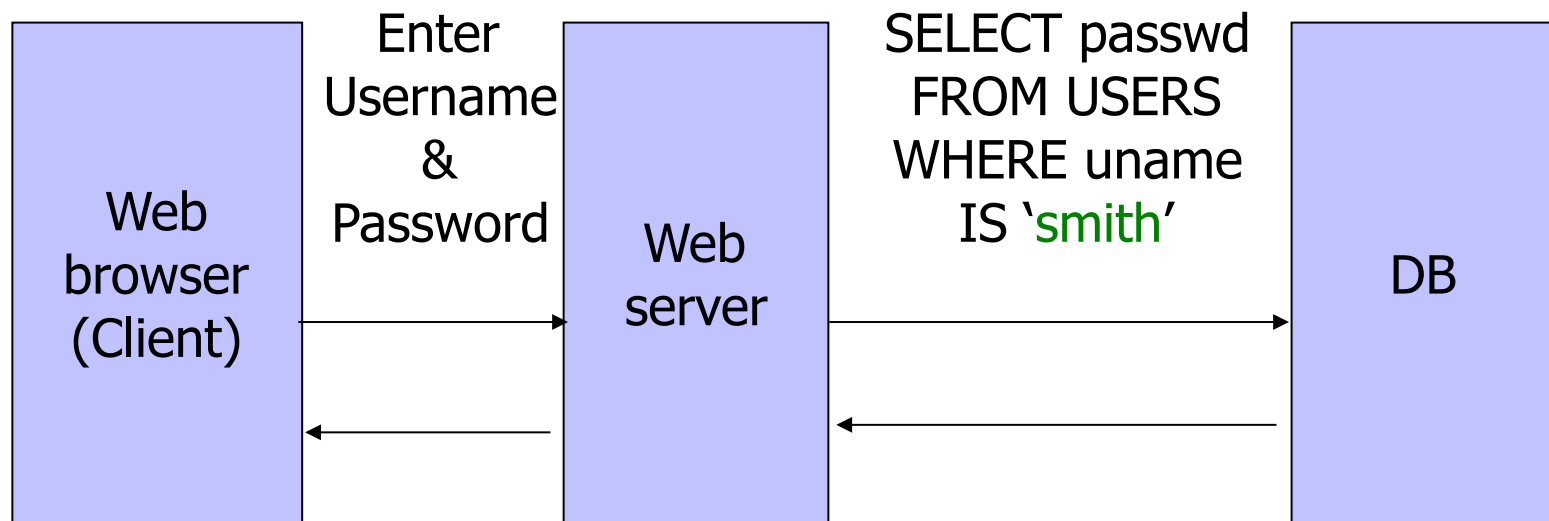
# User Input Becomes Part of Query

UNIVERSITY OF SAN FRANCISCO  
department of computer science





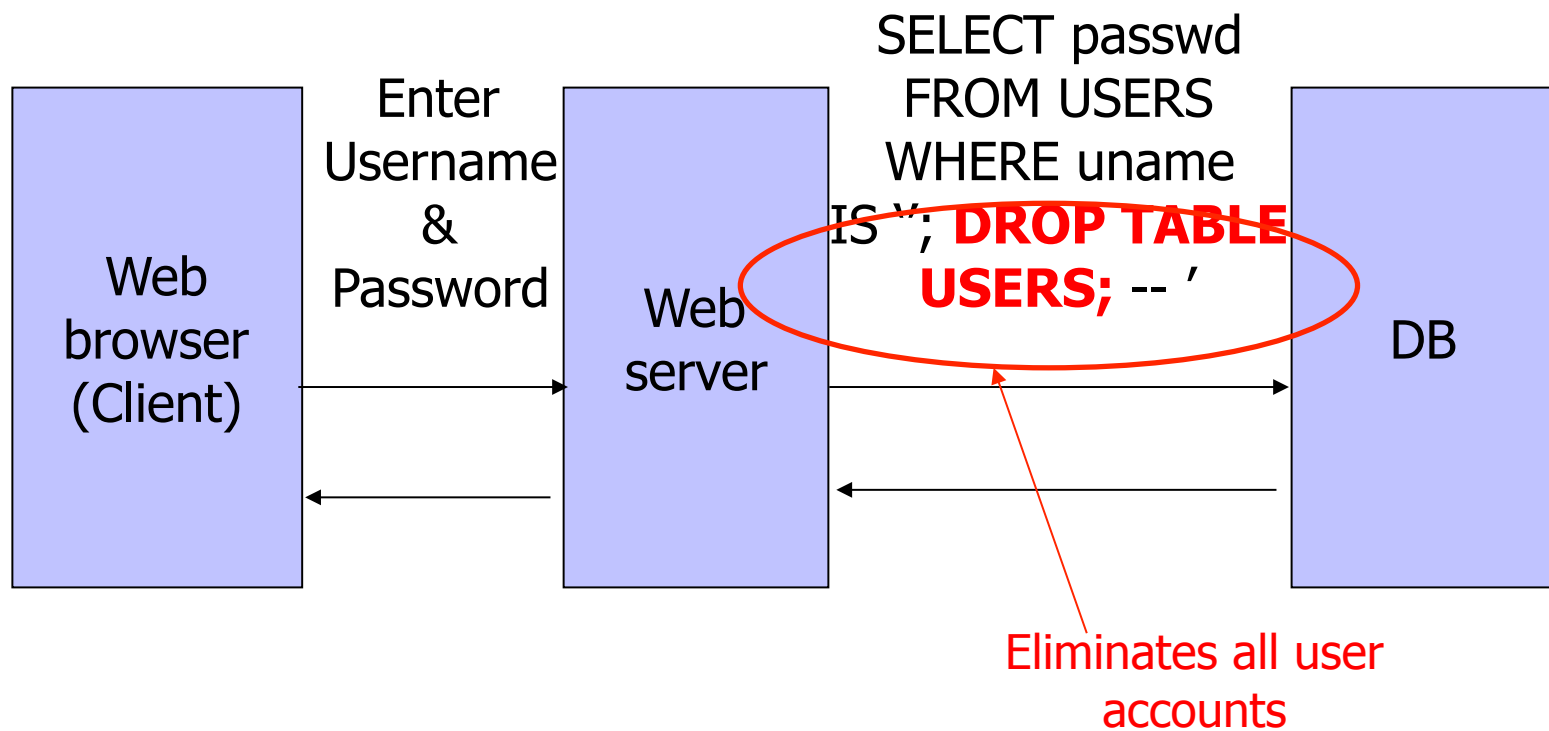
# Normal Login



# Malicious User Input

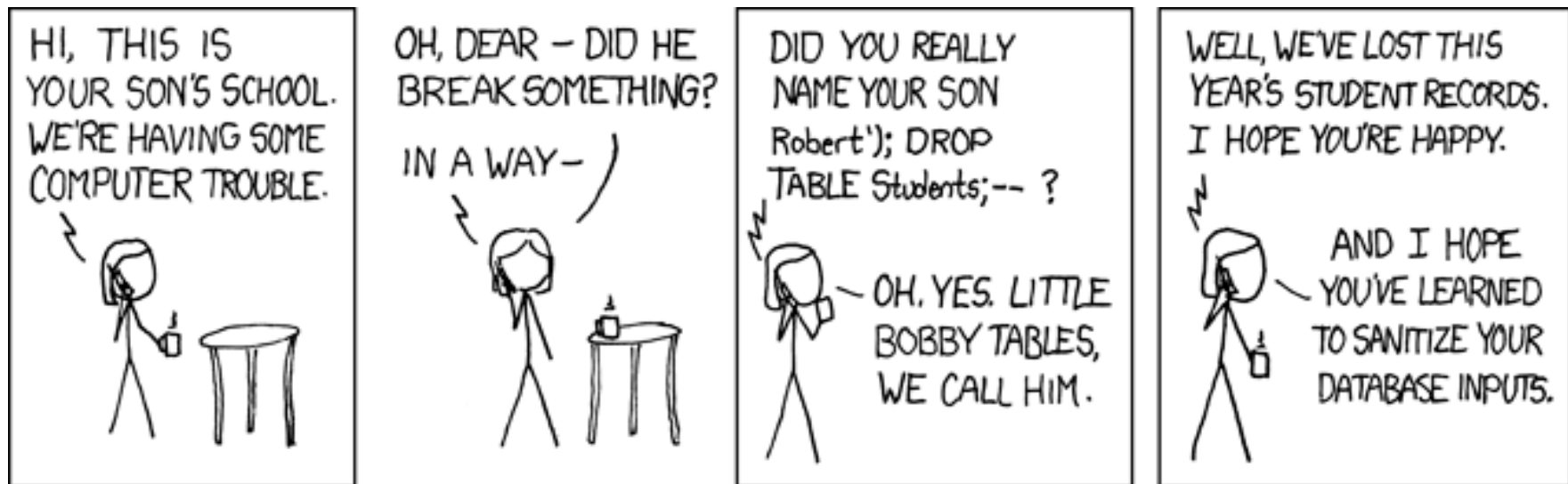


# SQL Injection Attack



# Exploits of a Mom

<http://xkcd.com/327/>



# 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

UNIVERSITY OF SAN FRANCISCO  
department of computer science

➤ User gives username ' OR 1=1 --

➤ Web server executes query

```
set UserFound=execute(  
    SELECT * FROM UserTable WHERE  
    username=' OR 1=1 -- ... );
```

Always true!

Everything after -- is ignored!

- Now all records match the query

➤ This returns the entire database!

# Another SQL Injection Example

[From Kevin Mitnick's "The Art of Intrusion"]

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

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

- User enters ' OR WHERE pwd LIKE '%' as both name and passwd

- Server executes

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

Wildcard matches any password

- Logs in with the credentials of the first person in the database (typically, administrator!)

# It Gets Better

- User gives username

' exec cmdshell 'net user badguy badpwd' / ADD --

- Web server executes query

```
set UserFound=execute(  
    SELECT * FROM UserTable WHERE  
    username= ' exec ... -- ... );
```

- 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))
```

Creates a password with 8  
random characters, **assuming**  
**\$new\_pass is set to NULL**

...

```
$result = dbquery("UPDATE ".$db_prefix."users  
    SET user_password=md5('$new_pass')  
    WHERE user_id='".$data['user_id']."'");
```

SQL query setting  
password in the DB

In normal execution, this becomes

```
UPDATE users SET user_password=md5('????????')  
WHERE user_id='userid'
```

User appends this to the URL:

`&new_pass=badPwd%27%29%2c`

`user_level=%27103%27%2cuser_aim=%28%27`

This sets \$new\_pass to  
`badPwd'), user_level='103', user_aim=('`

SQL query becomes

`UPDATE users SET user_password=md5('badPwd'),`

`user_level='103', user_aim=('??????')`

`WHERE user_id='user d'`

... with superuser privileges

User's password is  
set to 'badPwd'

# 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' --'`      **why does this work?**
- Solution: treat all parameters as dangerous

# SQL Injection in the Real World (1)

UNIVERSITY OF SAN FRANCISCO  
department of computer science

<http://www.ireport.com/docs/DOC-11831>

- 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"



45-35

# SQL Injection in the Real World (2)

UNIVERSITY OF SAN FRANCISCO  
department of computer science

- 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.



24-21

# 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





# April 2008 Attacks

UNIVERSITY OF SAN FRANCISCO  
department of computer science



The screenshot shows a web browser window with a blue title bar. The address bar contains "Live Search". The page header features a banner for "SECURITY FIX" with a photo of Brian Krebs and the text "BRIAN KREBS". Below the banner, the text "Brian Krebs on Computer Security" is visible. The main content area has a title "Hundreds of Thousands of Microsoft Web Servers Hacked" and a sub-header "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 article text discusses a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers, mentioning a patch issued last week and a Microsoft advisory (951306). It also mentions that the attacks are not related to IIS technologies and that SQL injection attacks can be used to install malicious software. The article concludes by mentioning that Shadowserver.org has a nice writeup and that the SANS Internet Storm Center also has information.

**SECURITY FIX** BRIAN KREBS

Brian Krebs on Computer Security

[About This Blog](#) | [Archives](#) | [XML RSS Feed](#) (What's RSS?)

## 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](#).

Microsoft Web Servers Hacked... Live Search

Done Internet 100%

# 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

# Part of the SQL Attack String

```
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

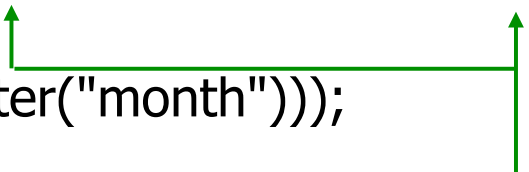
# Prepared Statements

- 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.

# Prepared Statement: Example

<http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html>

```
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();
```



Bind variable:  
data placeholder

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

# 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