CS 345

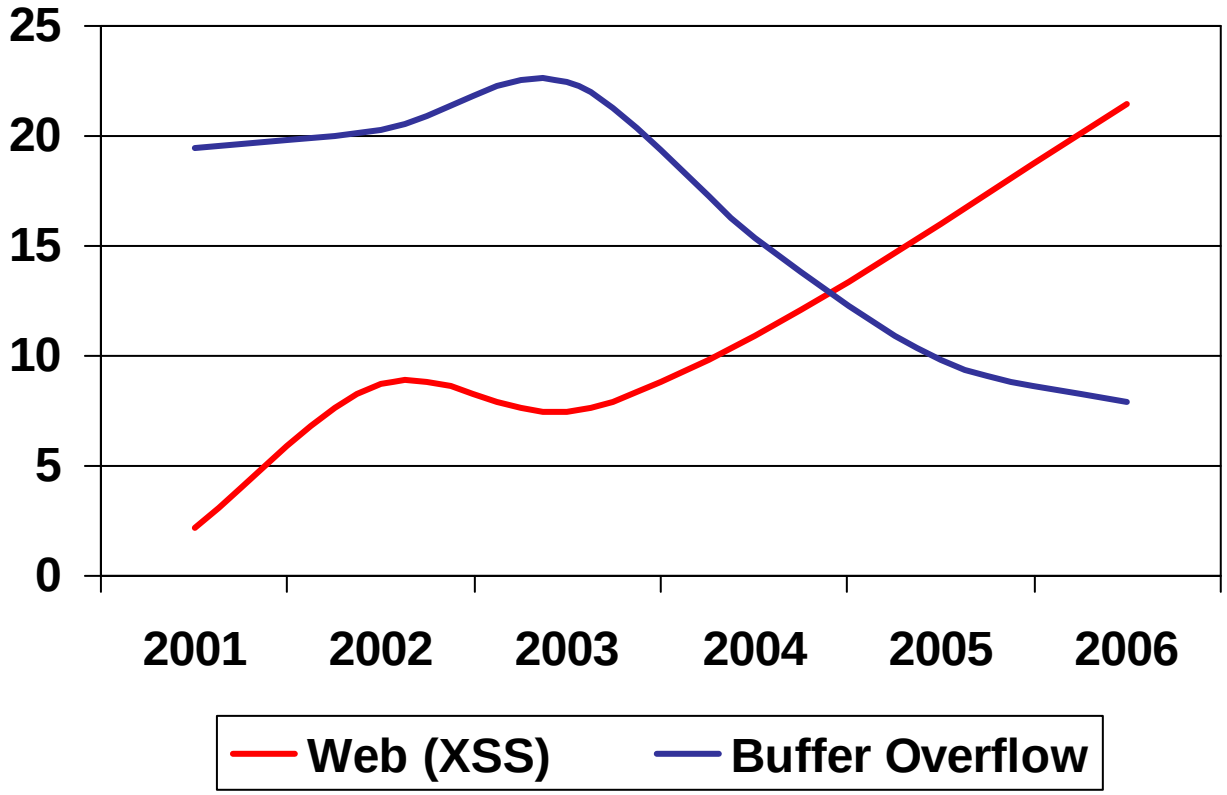
Security of Web Applications

Vitaly Shmatikov

Vulnerability Stats: Web is "Winning"

Source: MITRE CVE trends

Majority of vulnerabilities now found in web software



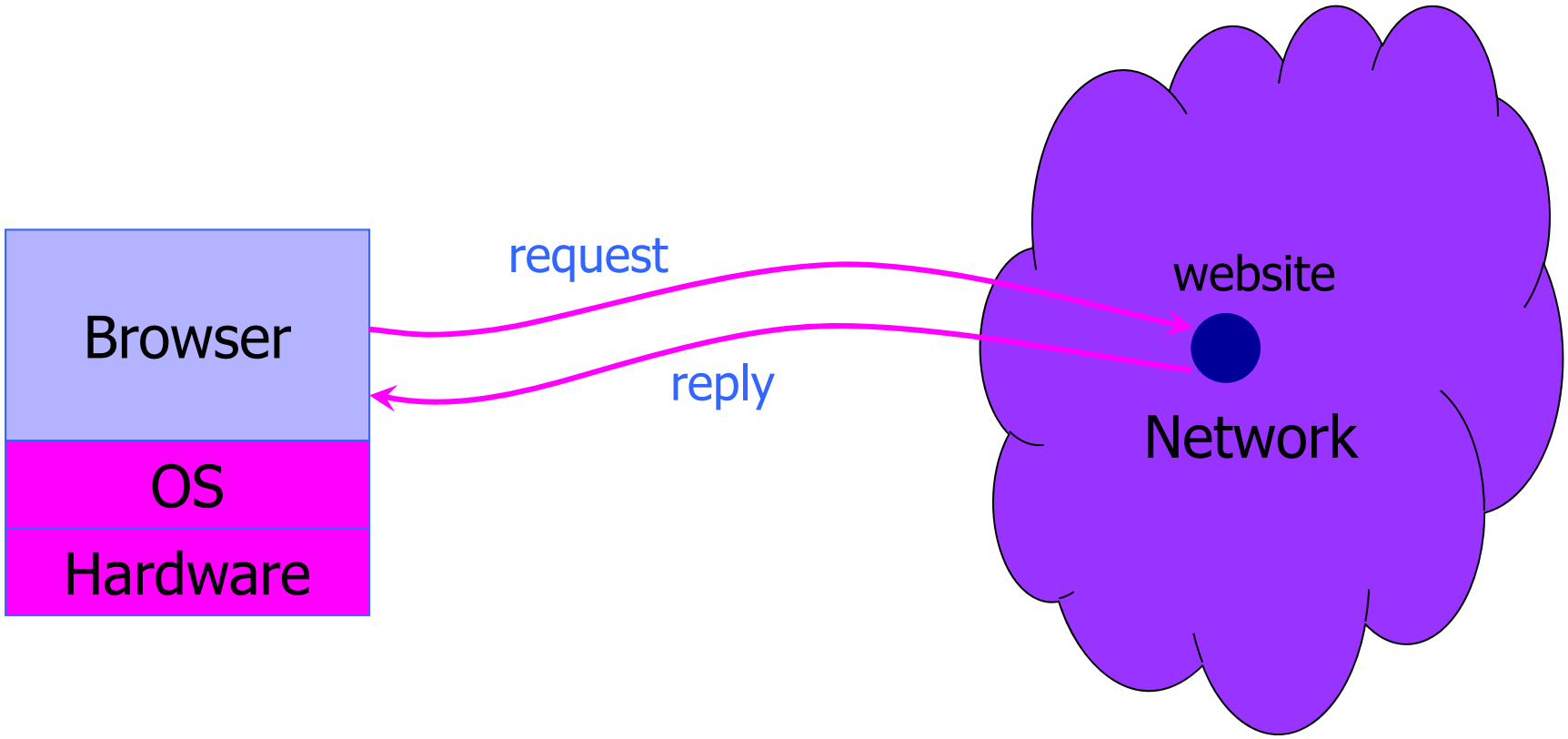
Web Applications

- Big trend: software as a (Web-based) service
  - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  - Cloud computing
- Applications hosted on Web servers
  - Written in a mixture of PHP, Java, Perl, Python, C, ASP
  - Poorly written scripts with inadequate input validation

Typical Web Application Design

- Runs on a Web server or application server
- Takes input from Web users (via Web server)
- Interacts with back-end databases and third parties
- Prepares and outputs results for users (via Web server)
  - Dynamically generated HTML pages
  - Contain content from many different sources, often including regular users
    - Blogs, social networks, photo-sharing websites…

Browser and Network



Two Sides of Web Applications

- Web browser
  - Executes JavaScript presented by websites the user visits
- Web application
  - Runs at website
    - Banks, online merchants, blogs, Google Apps, many others
  - Written in PHP, ASP, JSP, Ruby, …

JavaScript Security Model

- Script runs in a "sandbox"
  - No direct file access, restricted network access
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other

Library Import

- Same-origin policy does not apply to scripts loaded in enclosing frame from arbitrary site

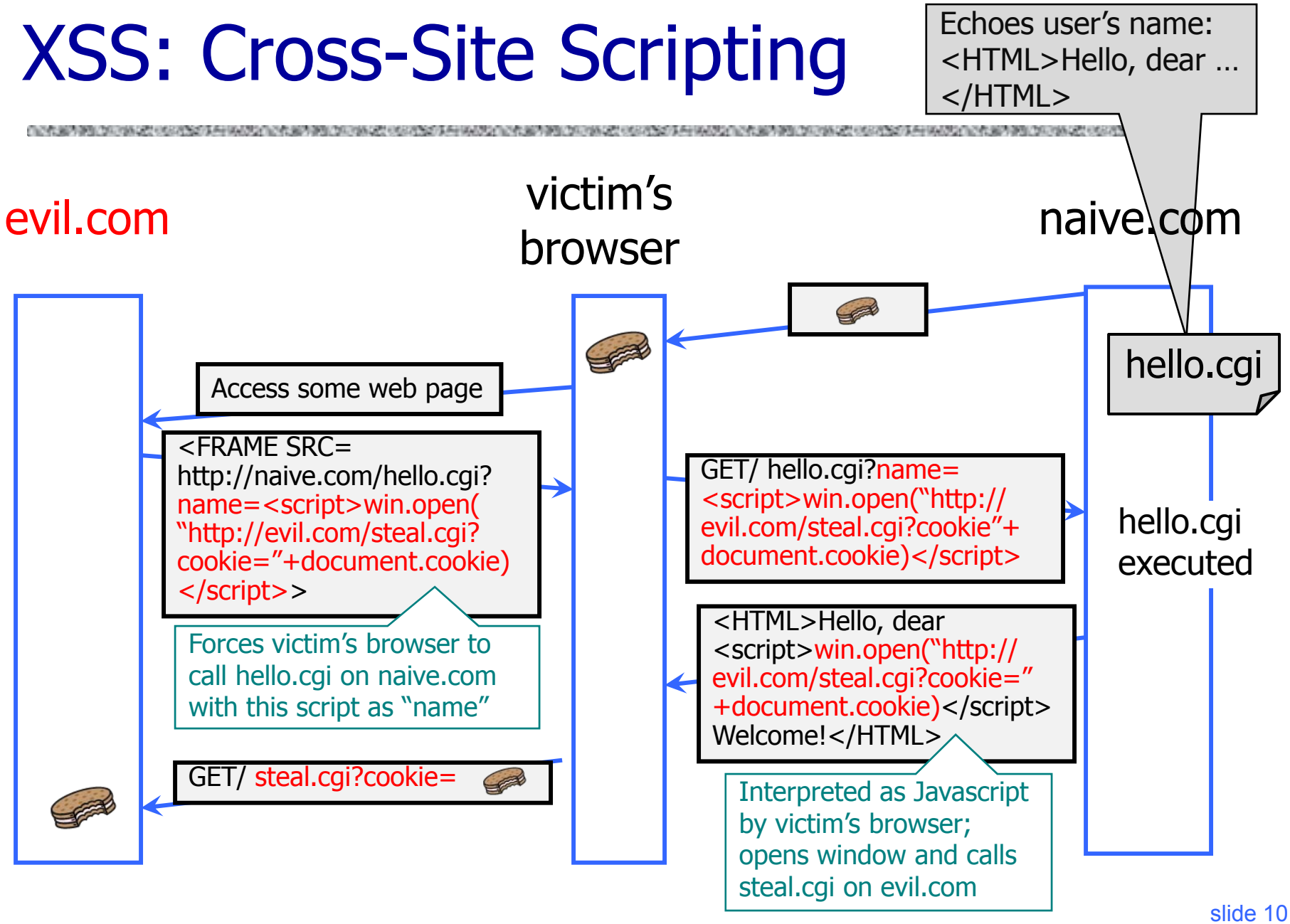
```
<script type="text/javascript">
    src="http://www.example.com/scripts/somescript.js">
</script>
```

- This script runs as if it were loaded from the site that provided the page!

Web Attacker

- Controls malicious website (attacker.com)
  - Can even obtain SSL/TLS certificate for his site ($0)
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by ad network, blind luck …
- Attacker has no other access to user machine!
- Variation: gadget attacker
  - Bad gadget included in otherwise honest mashup (EvilMaps.com)

XSS: Cross-Site Scripting



So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g., Gmail)
  - Link in DoubleClick banner ad
  - … many many ways to fool user into clicking
- So what if evil.com gets cookie for naive.com?
  - Cookie can include session authenticator for naive.com
    - Or other data intended only for naive.com
  - Violates the "intent" of the same-origin policy

Other XSS Risks

- XSS is a form of "reflection attack"
  - User is tricked into visiting a badly written website
  - A bug in website code causes it to display and the user's browser to execute an arbitrary attack script
- Can change contents of the affected website by manipulating DOM components
  - Show bogus information, request sensitive data
  - Control form fields on this page and linked pages
    - For example, MySpace.com phishing attack injects password field that sends password to bad guy
- Can cause user's browser to attack other websites

Where Malicious Scripts Lurk

- Hidden in user-created content
  - Social sites (e.g., MySpace), blogs, forums, wikis
- When visitor loads the page, webserver displays the content and visitor's browser executes script
  - Many sites try to filter out scripts from user content, but this is difficult

MySpace Worm (1)

http://namb.la/popular/tech.html

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
  - No <script>, <body>, onclick, <a href=javascript://>
- … but does allow tags for CSS. K00L!
  -
- 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)

http://namb.la/popular/tech.html

- *"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 piss anyone off. 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

XSS in Orkut

http://antrix.net/journal/techtalk/orkut_xss.html

Example of XSS exploit code

- Orkut: Google's social network
  - 37 million members (2006), very popular in Brazil
- Bug allowed users to insert scripts in their profiles
- Orkut Cookie Exploit: user views infected profile, all groups he owns are transferred to attacker
- virus.js: attack script in a flash file
  - Every viewer of infected profile is joined to a community
    - "Infectatos pelo Virus do Orkut" (655,000 members at peak!)
  - Virus adds malicious flash as a "scrap" to the visitor's profile; everybody who views that profile is infected, too
    - Exponential propagation!

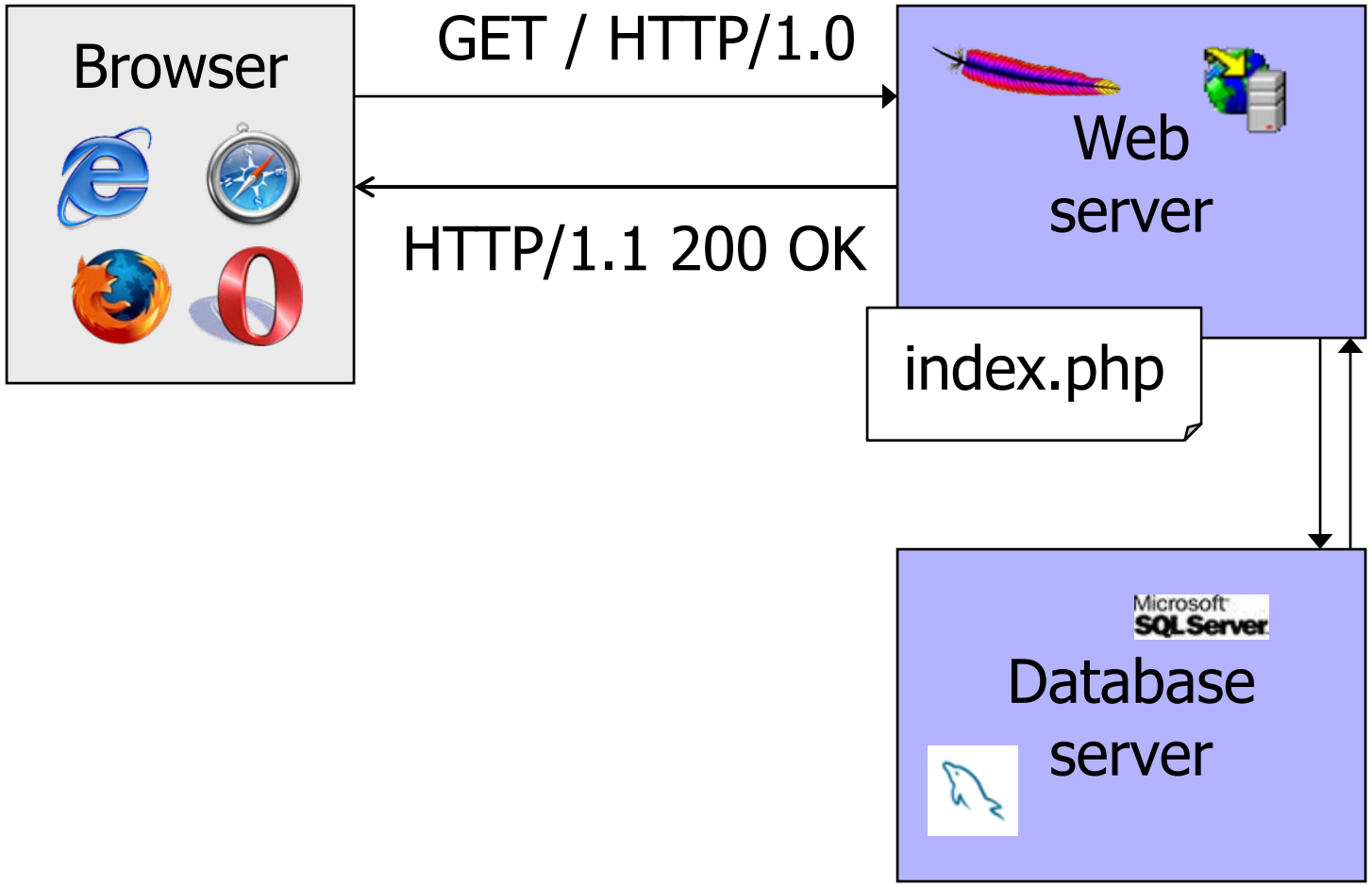
Similar to "wall post" in Facebook

Preventing Cross-Site Scripting

- Preventing injection of scripts into HTML is hard!
  - Blocking "<" and ">" is not enough
  - Event handlers, stylesheets, encoded inputs (%3C), etc.
  - phpBB allowed simple HTML tags like <b>

    <b c=">" onmouseover="script" x="<b ">Hello<b>
- Any user input must be preprocessed before it is used inside HTML
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes ' " becomes " & becomes &
  - In ASP.NET, Server.HtmlEncode(string)

Dynamic Web Applications



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, ...

SQL

- 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

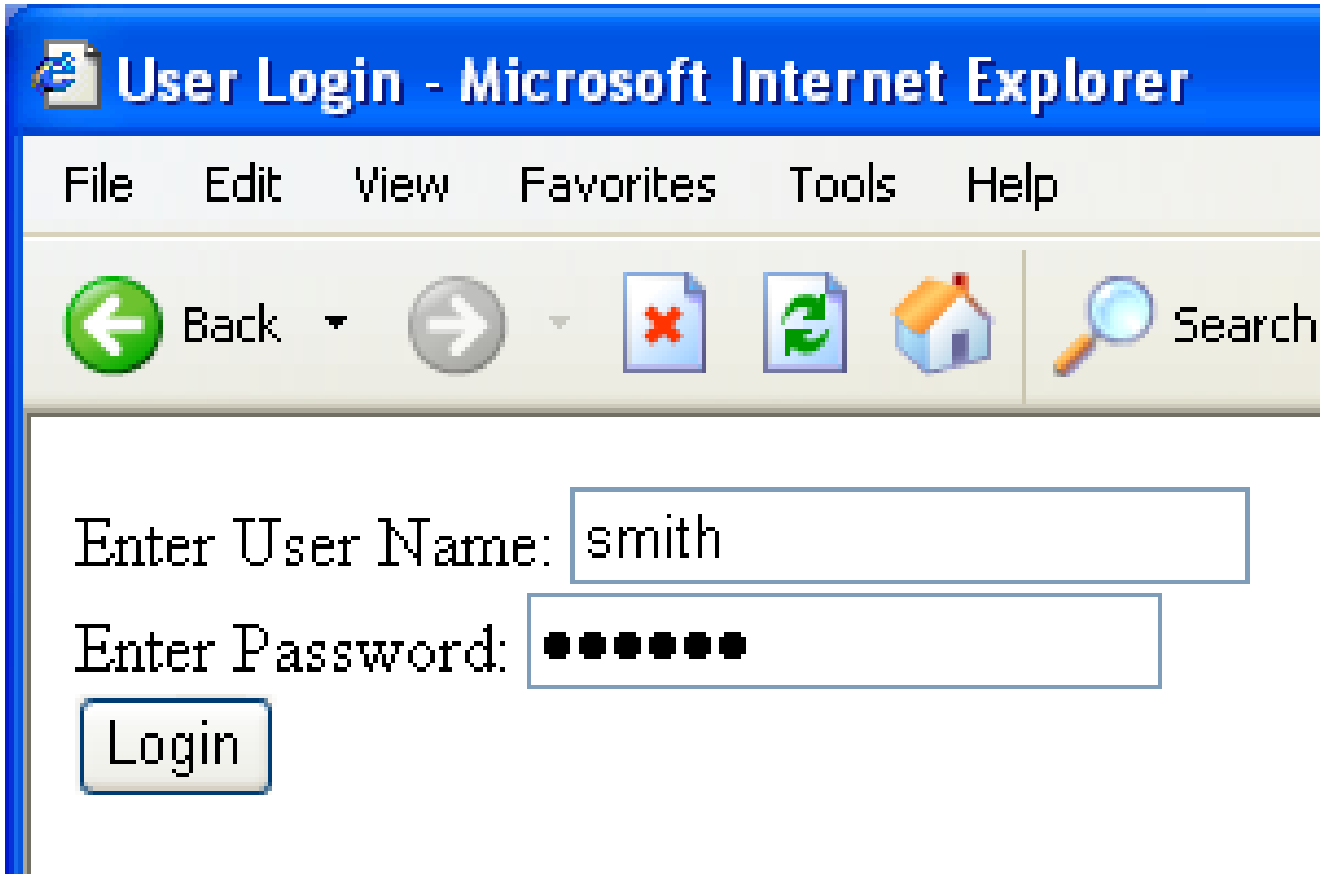
Sample Code

- Sample PHP

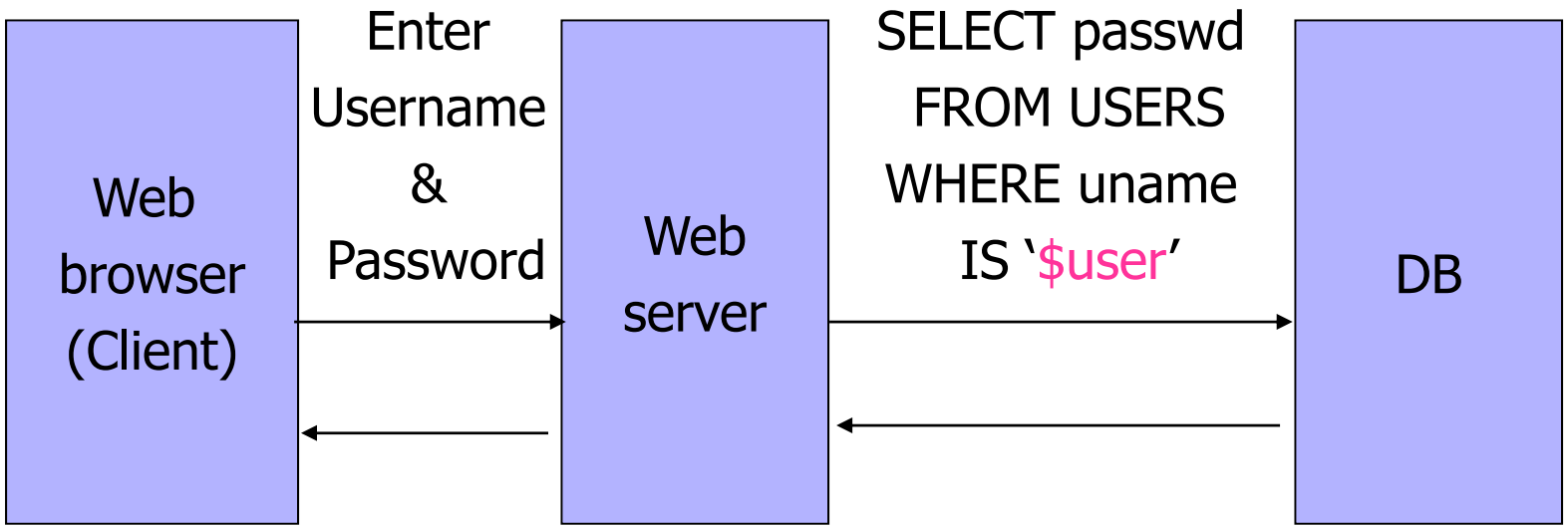
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
          "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

- What if ‘user’ is a malicious string that changes the meaning of the query?

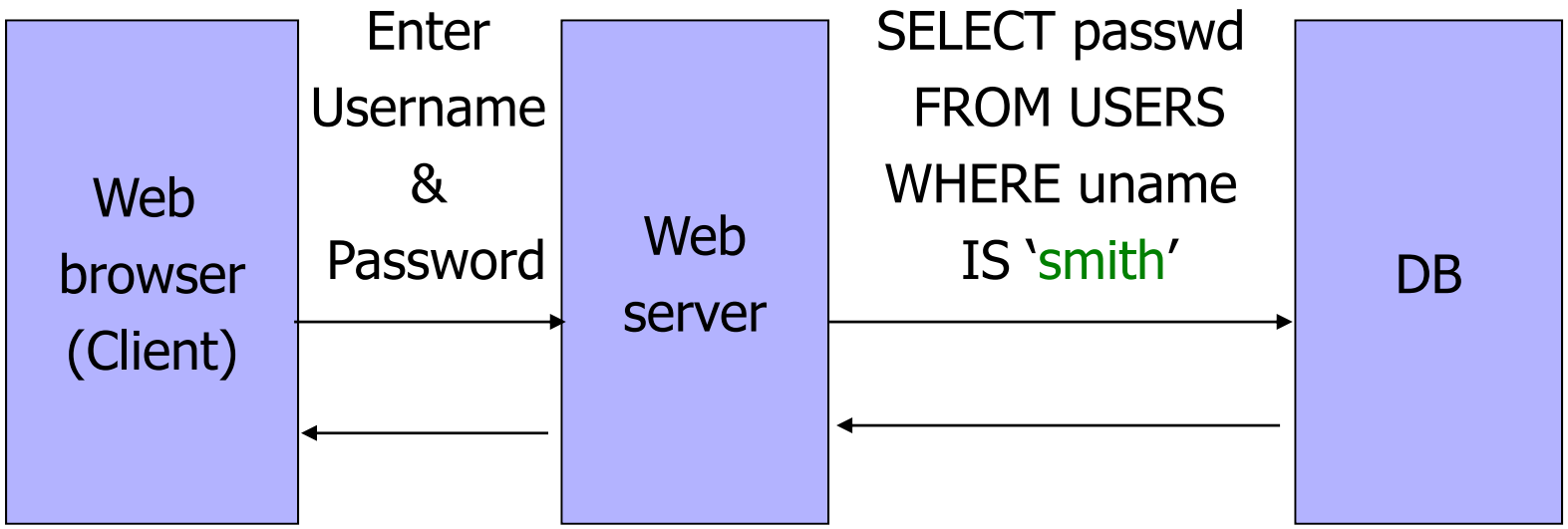
Typical Login Prompt



User Input Becomes Part of Query

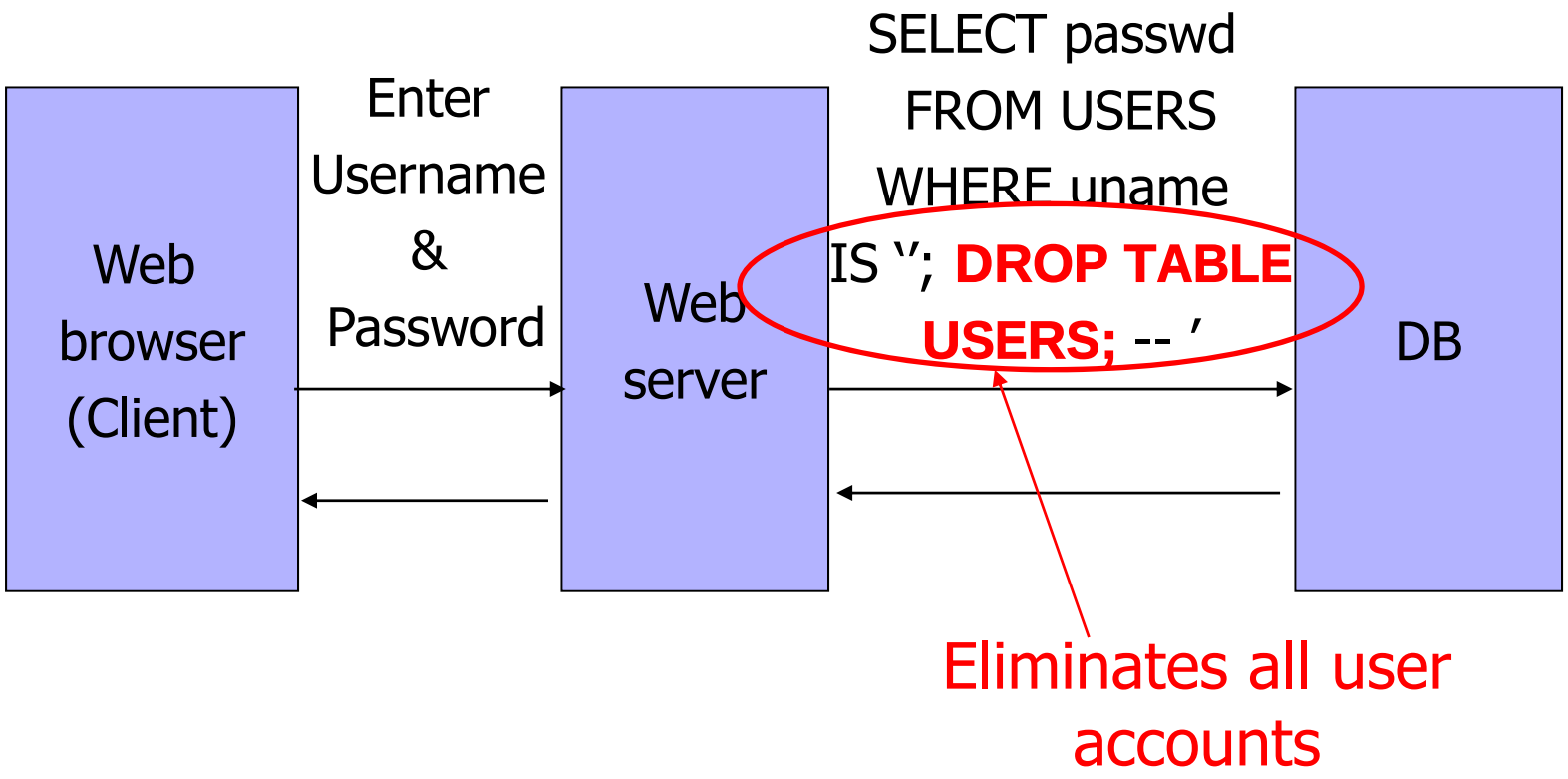


Normal Login



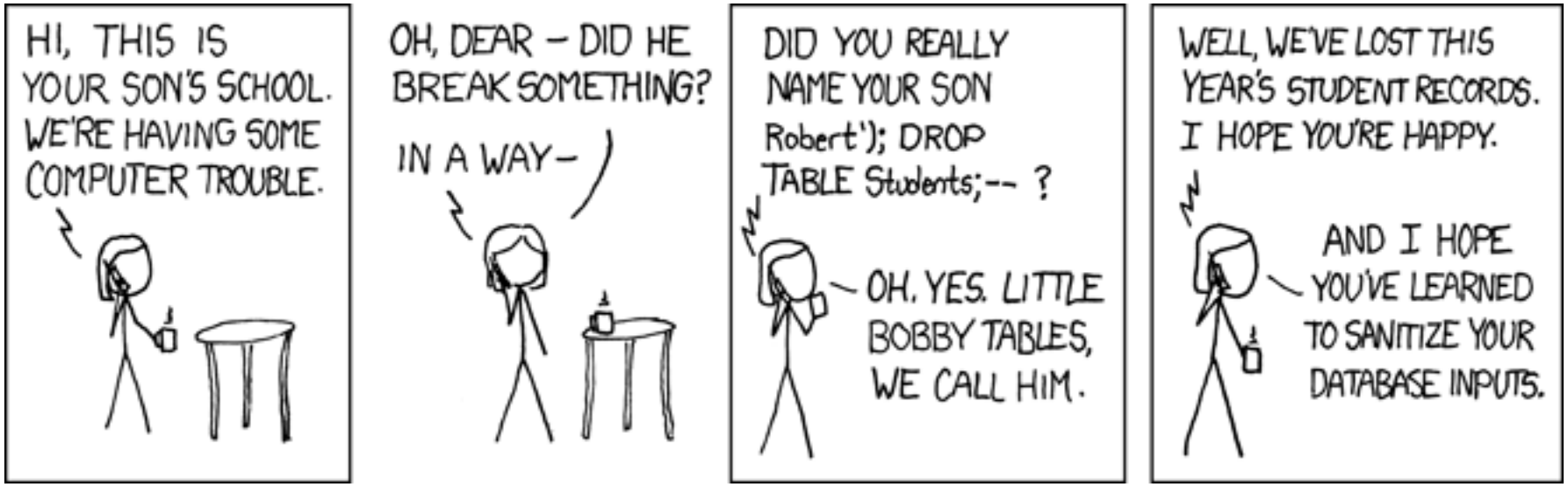
Malicious User Input

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

- 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!

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']." ' ");
```

Creates a password with 8 random characters, assuming $new_pass is set to NULL

SQL query setting password in the DB

In normal execution, this becomes

```
UPDATE users SET user_password=md5('&5h!@*r5')
WHERE user_id='userid'
```

Exploit

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=('&5h!@*r5')
WHERE user_id='userid'
```

… with superuser privileges

User's password is set to 'badPwd'

SQL Injection in the Real World

- 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

Attack on Microsoft IIS (April 2008)

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.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was *not* the fault of a flaw in IIS: "..our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

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

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

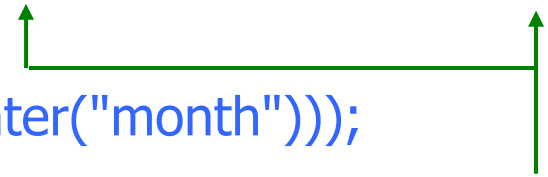
Prepared Statements

- Metacharacters such as ’ in queries provide distinction between data and code
- In most injection attacks data are interpreted as code – 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 → 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.getParamenter("month")));
ResultSet res = ps.executeQuery();
```



Bind variable:
data placeholder

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