CSE 484 / CSE M 584: Computer Security and Privacy

Web Security: Web Application Security [continued]

Fall 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- Reminder: Friday is a holiday
  - No class or office hours
- Lab 2
  - Sign up if you haven't!
  - Check forum post for whether you should have access (let us know if having trouble)
  - FYI: Running your sites on **homes.cs.washington.edu** is not a web security things but a restriction of the lab itself

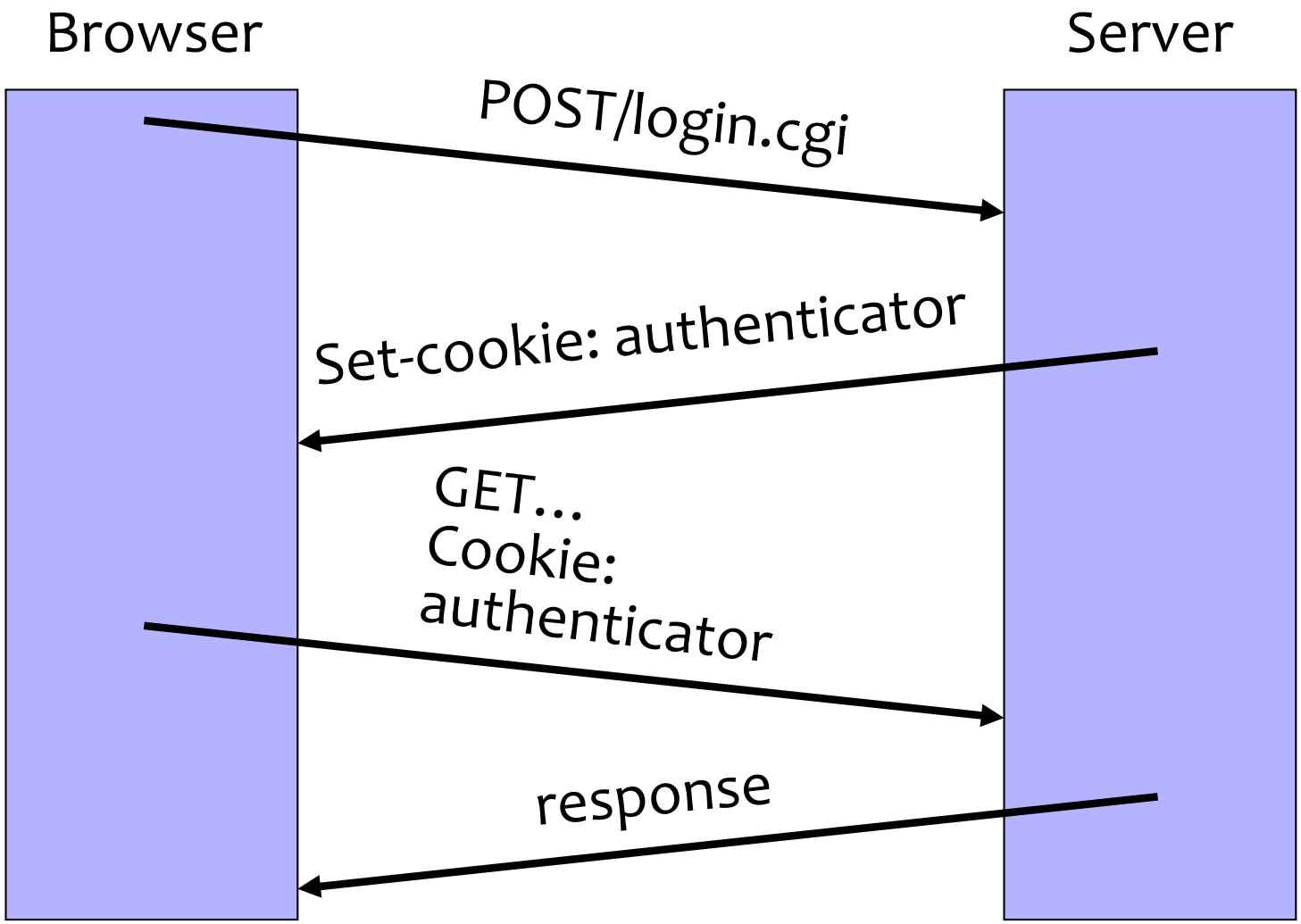
http://www.owasp.org

OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Redux



Browser Sandbox Redux

- Based on the same origin policy (SOP)
- Active content (scripts) can send anywhere!
  - For example, can submit a POST request
  - Some ports inaccessible -- e.g., SMTP (email)
- Can only *read* response from the *same origin*
  - … but you can do a lot with just sending!

Cross-Site Request Forgery

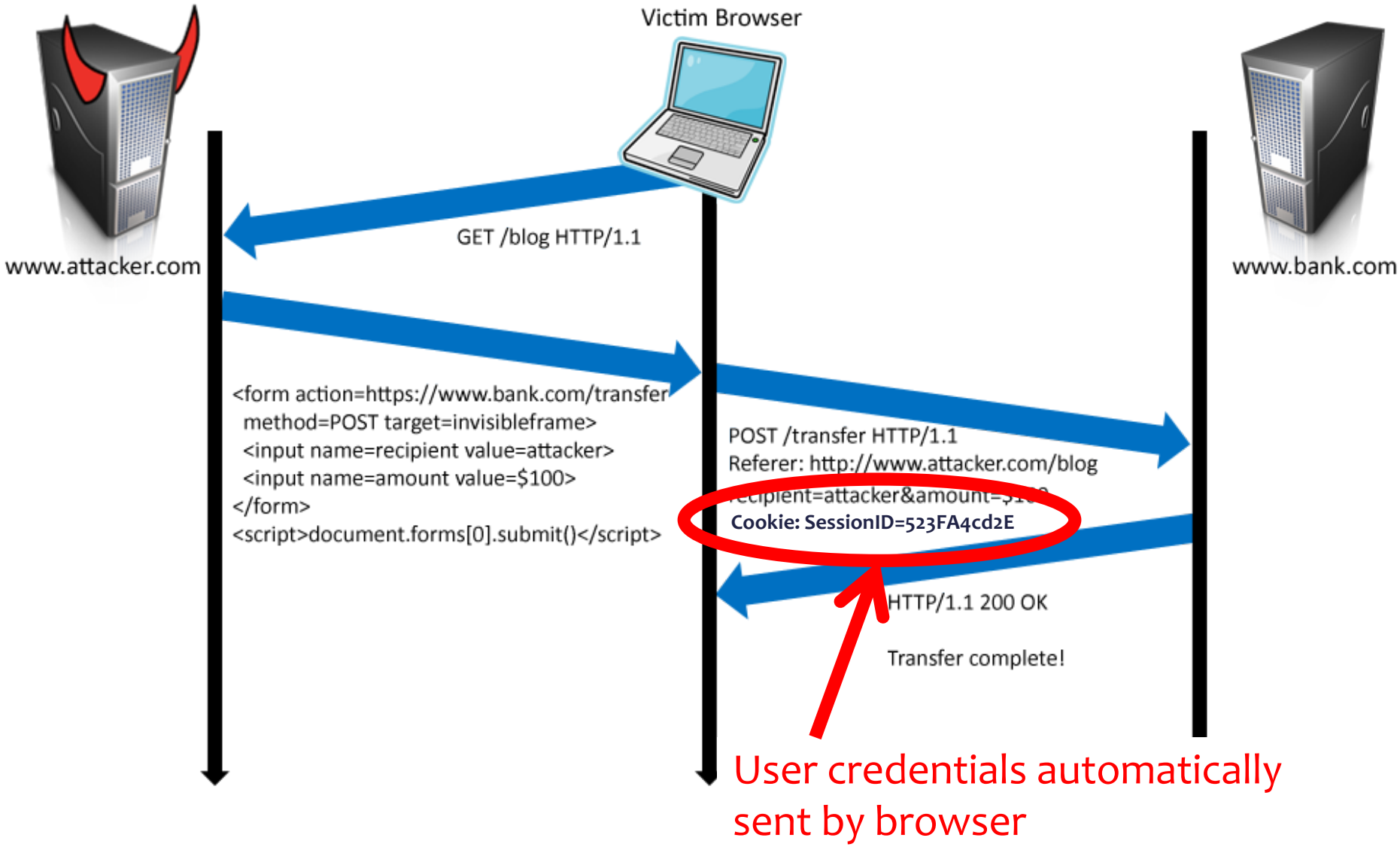
- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

```
<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!
- Lesson: cookie authentication is not sufficient when side effects can happen

Cookies in Forged Requests



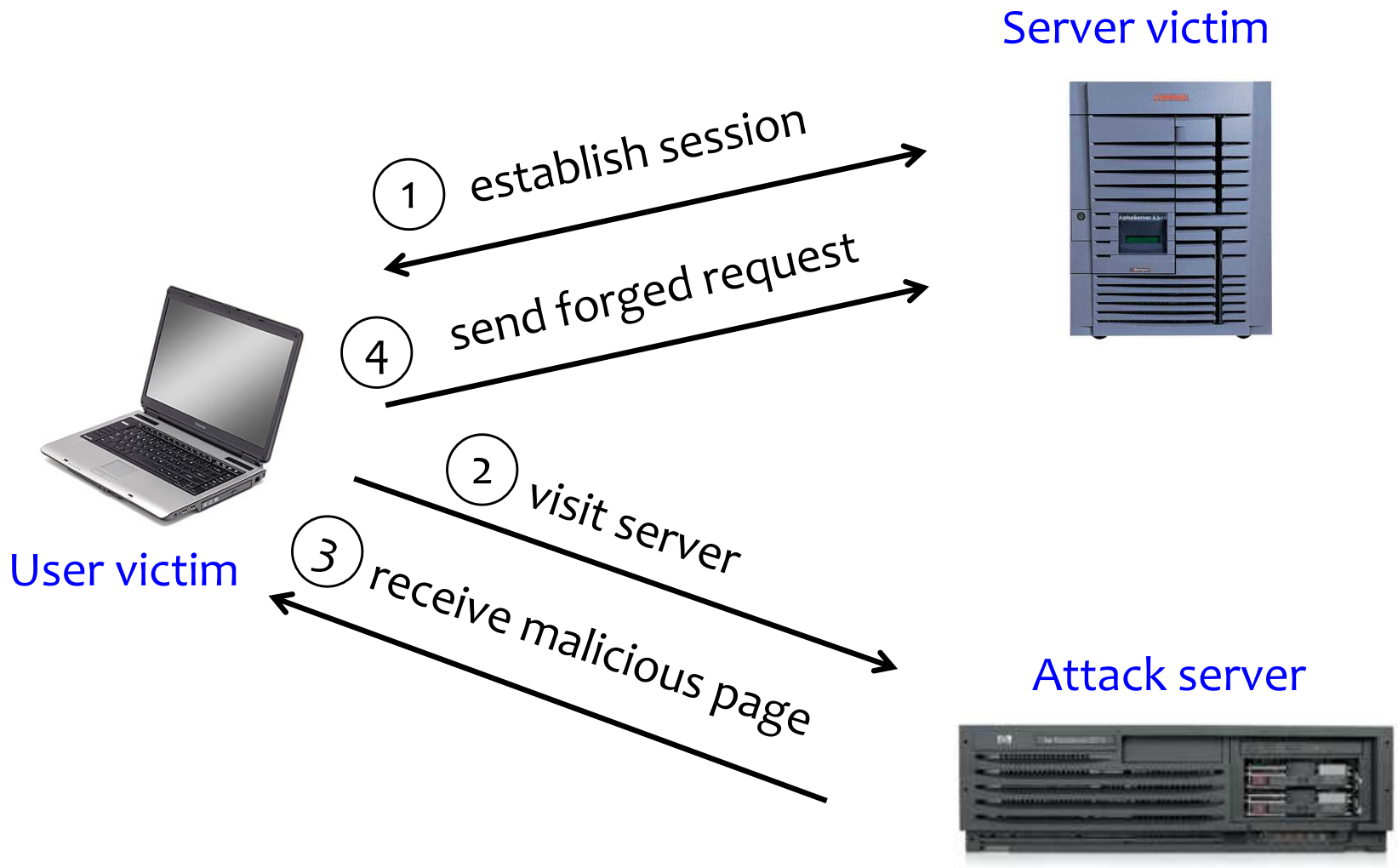
Sending a Cross-Domain POST

```
<form method="POST" action=http://othersite.com/action >
...
</form>
<script>document.forms[0].submit()</script>
```

submit post

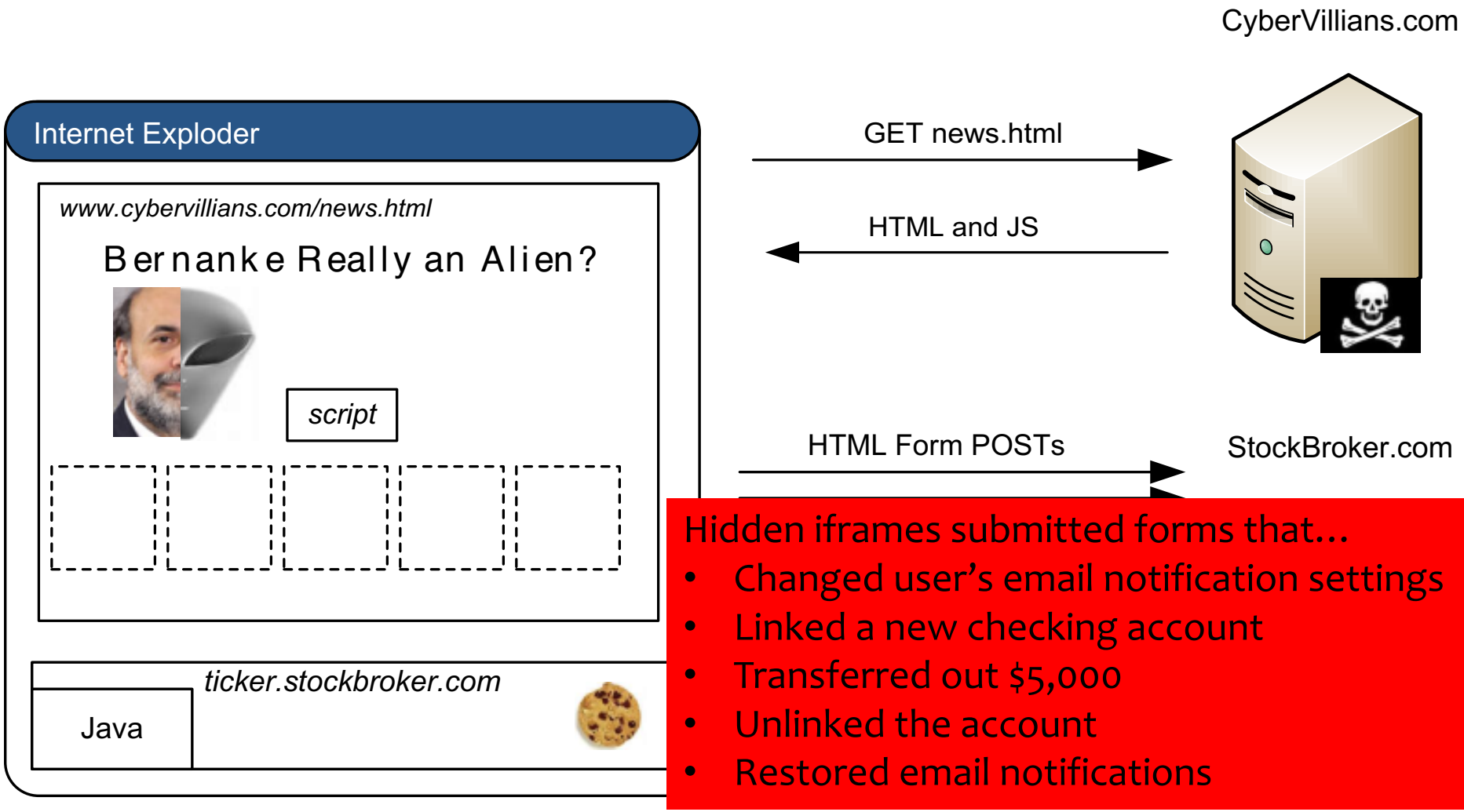
- Hidden iframe can do this in the background
- User visits a malicious page, browser submits form on behalf of the user
  - Hijack any ongoing session (if no protection)
    - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
  - Reprogram the user's home router
  - Many other attacks possible

XSRF (aka CSRF): Summary



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

XSRF True Story [Alex Stamos]



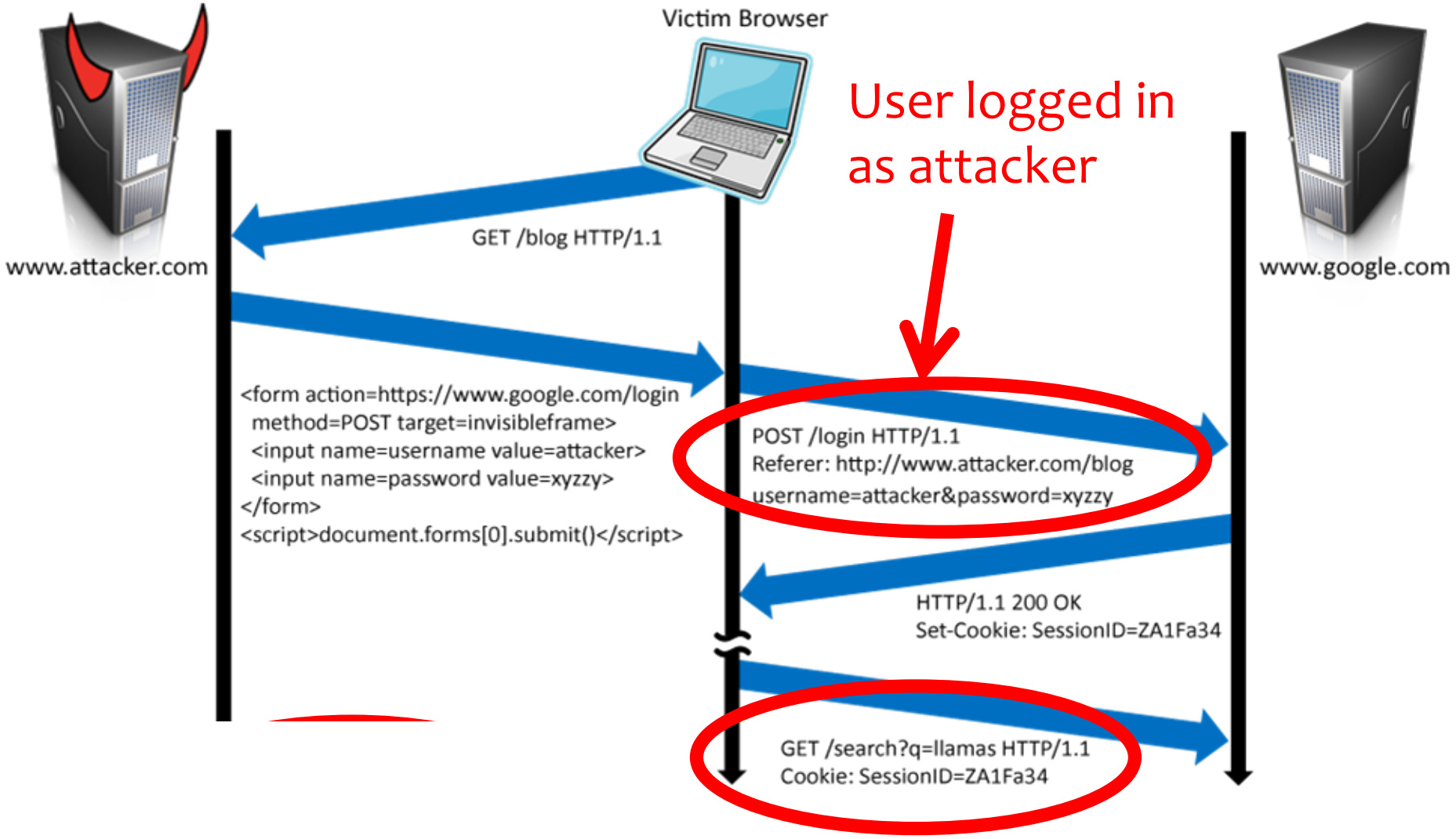
Hidden iframes submitted forms that…

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

Broader View of XSRF

- Abuse of cross-site data export
  - SOP does not control data export
  - Malicious webpage can initiates 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 (automatically sends cookies)

Login XSRF: Attacker logs you in as them!



XSRF Defenses

- Secret validation token


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

- Referer validation

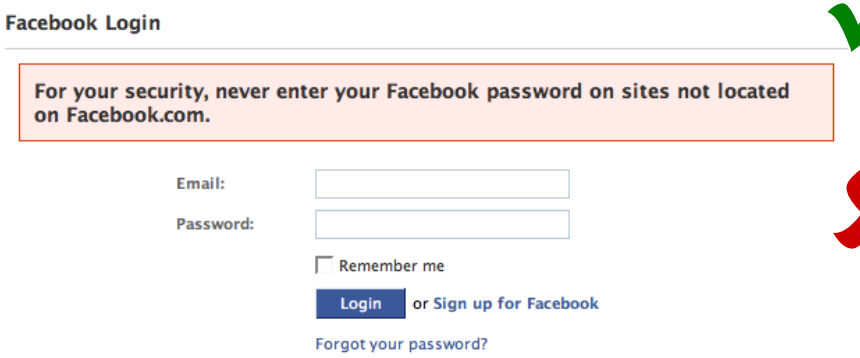

```
Referer:
http://www.facebook.com/home.php
```

Add Secret Token to Forms

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

- "Synchronizer Token Pattern"
- Include a secret challenge token as a hidden input in forms
  - Token often based on user's session ID
  - Server must verify correctness of token before executing sensitive operations
- Why does this work?
  - **Same-origin policy:** attacker can't read token out of legitimate forms loaded in user's browser, so can't create fake forms with correct token

Referer Validation



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

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

? `Referer:`

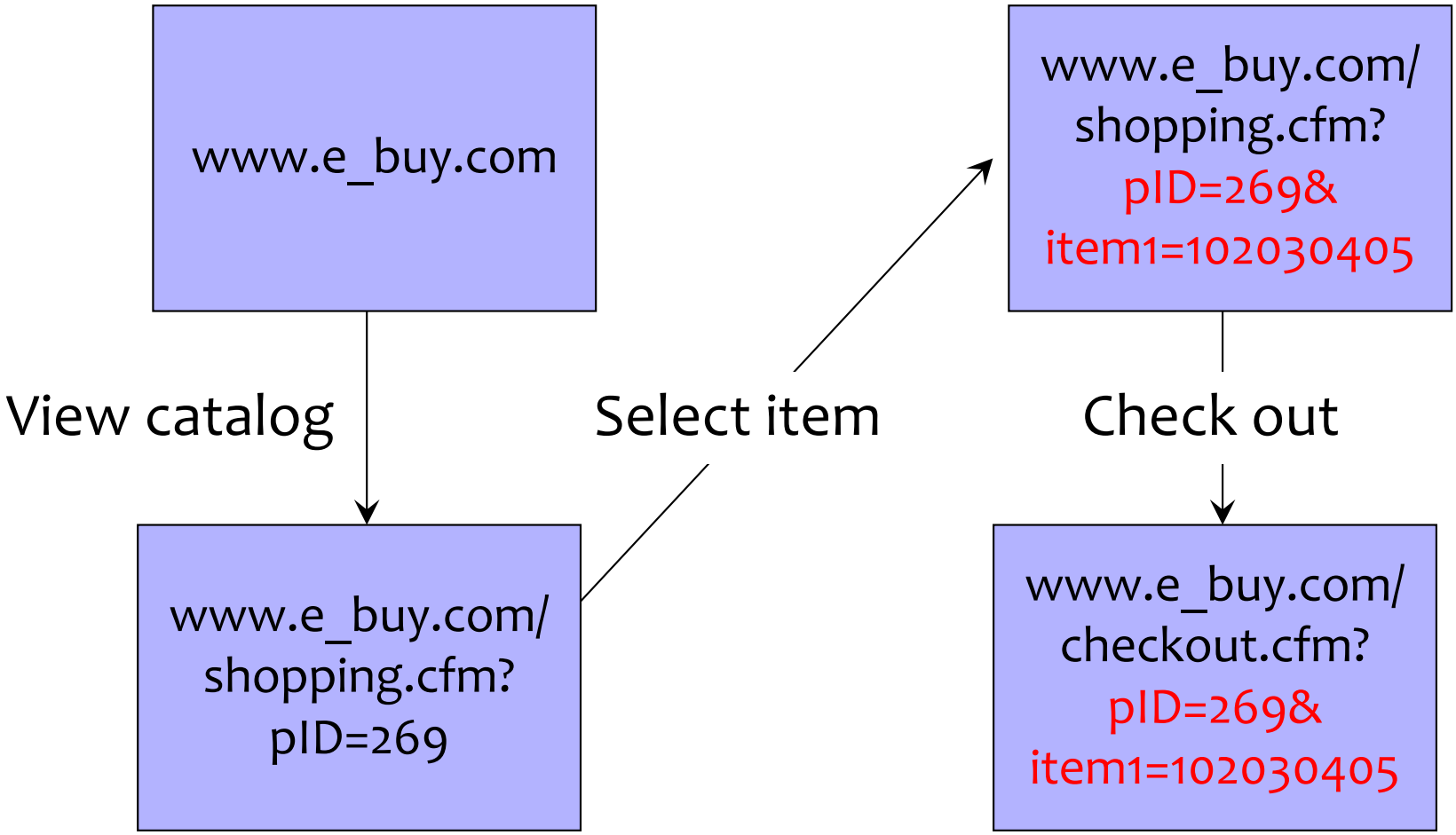
- Lenient referer checking – header is optional
- Strict referer checking – header is required

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

Web Session Management

Primitive Browser Session



Store session information in URL; easily read on network

Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping and modification
- There is no guarantee that URL is private

FatBrain.com circa 1999

- User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

  https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  - With special URL, user doesn't need to re-authenticate
    - Reasoning: user could not have not known the special URL without authenticating first. That's true, BUT…
- Authenticators are global sequence numbers
  - It's easy to guess sequence number for another user

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  - Partial fix: use random authenticators

Typical Solution: Web Authentication via Cookies

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticators must be **unforgeable** and **tamper-proof**
      - Malicious client shouldn't be able to compute his own or modify an existing authenticator
    - Example: **MAC(server's secret key, session id)**
  - With each request, browser presents the cookie
  - Server **recomputes** and verifies the authenticator
    - Server does not need to remember the authenticator

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="B
    with leather straps">

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

</FORM>
```

Change this to 2.00

Bargain shopping!

Fix: MAC client-side data, or, more likely, keep on server.

Top Web Vulnerabilities: Summary

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