CSE 484 / CSE M 584: Computer Security and Privacy

Web Security: Web Application Security [continued]

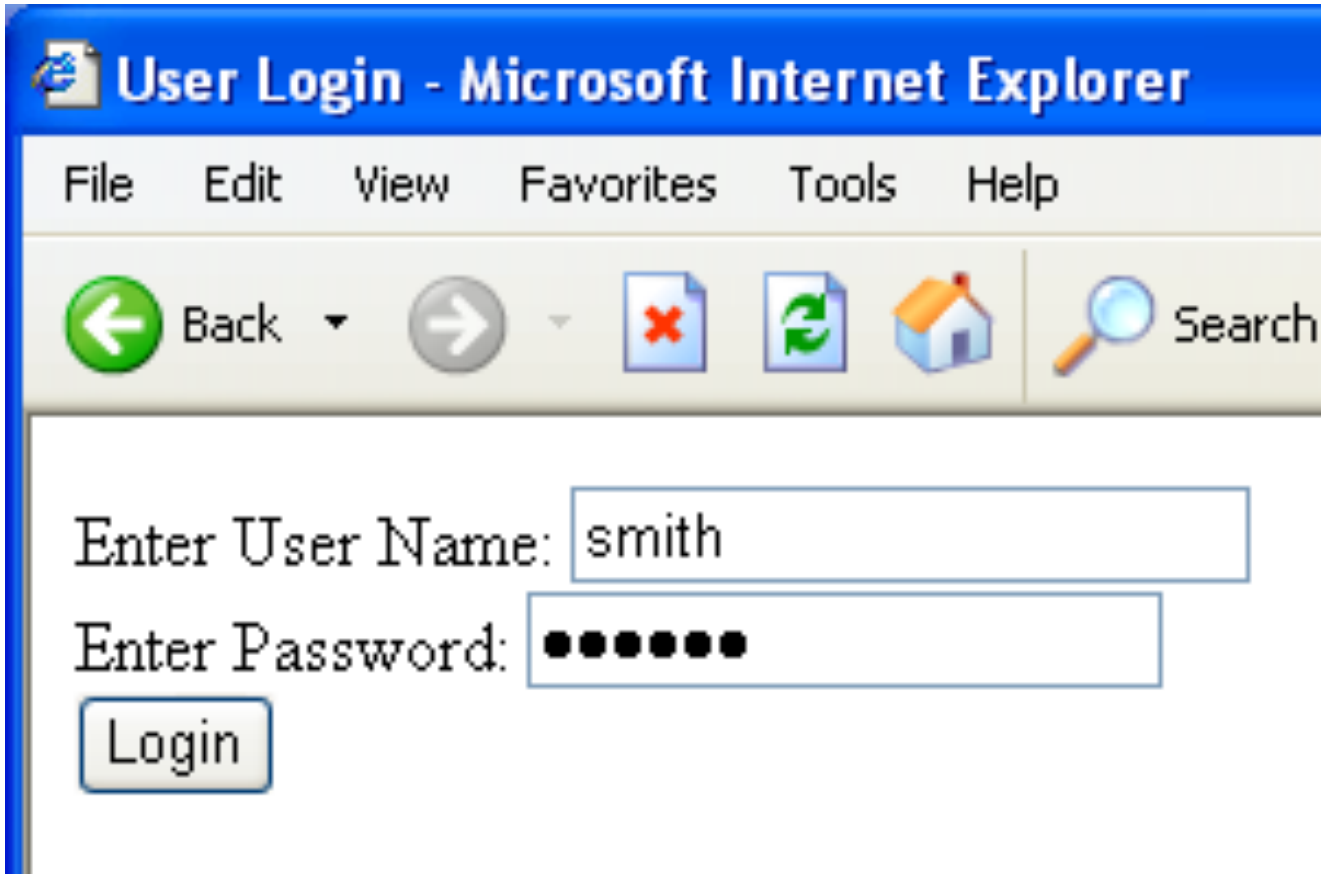
Spring 2017

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

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

SQL Injection

Typical Login Prompt

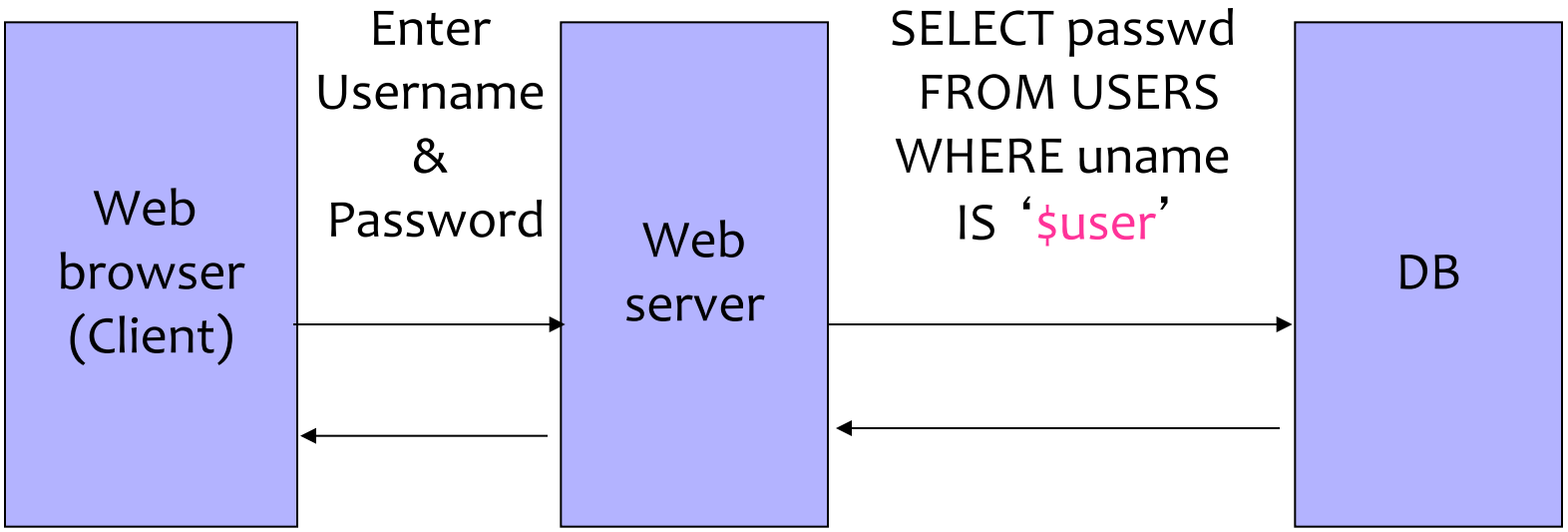


Typical Query Generation Code

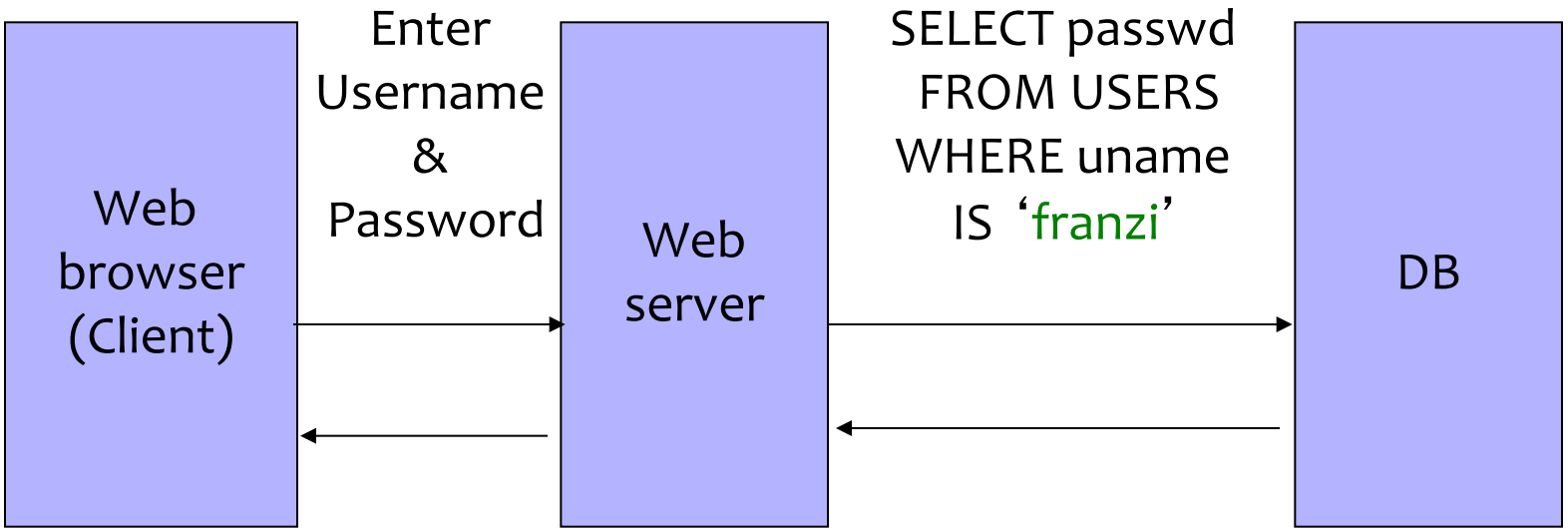
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
       "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

What if **'user'** is a malicious string that changes the meaning of the query?

User Input Becomes Part of Query

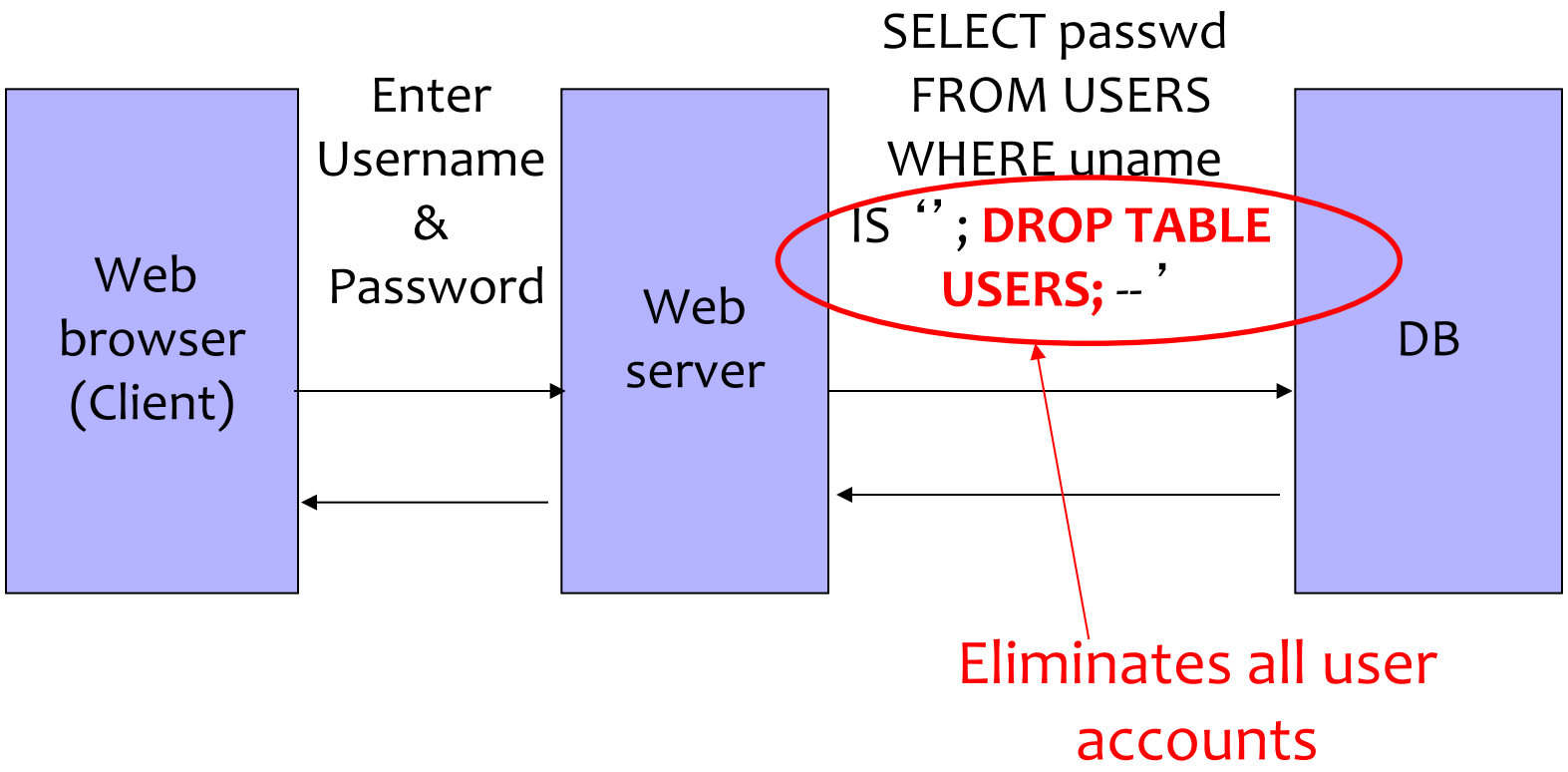


Normal Login

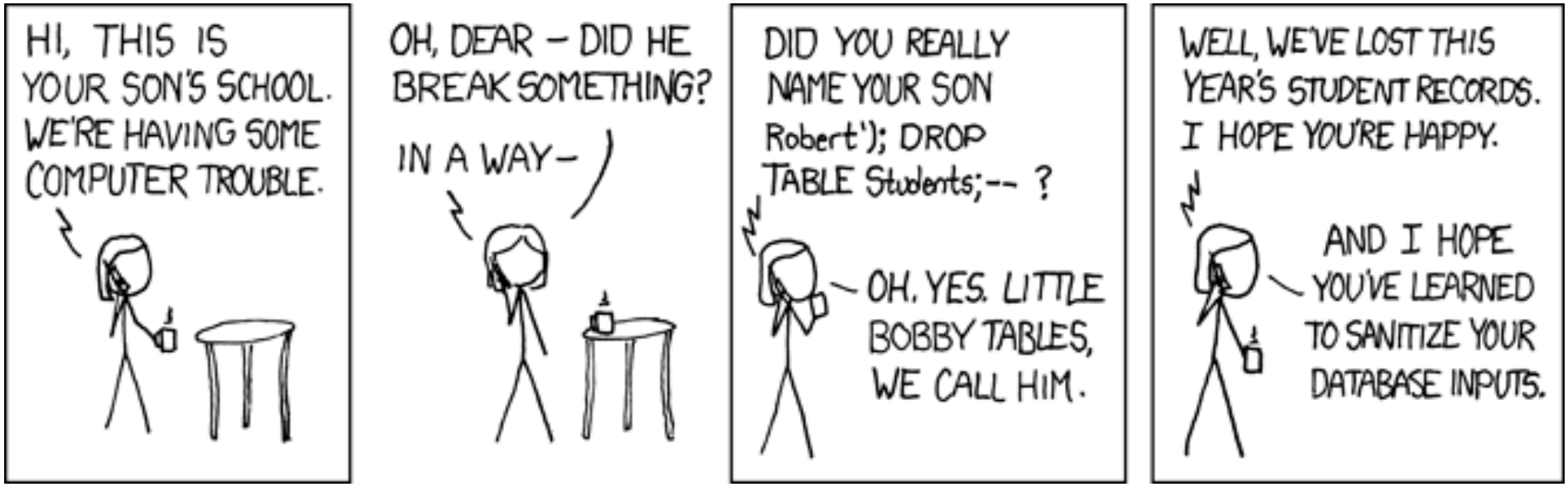


Malicious User Input

SQL Injection Attack

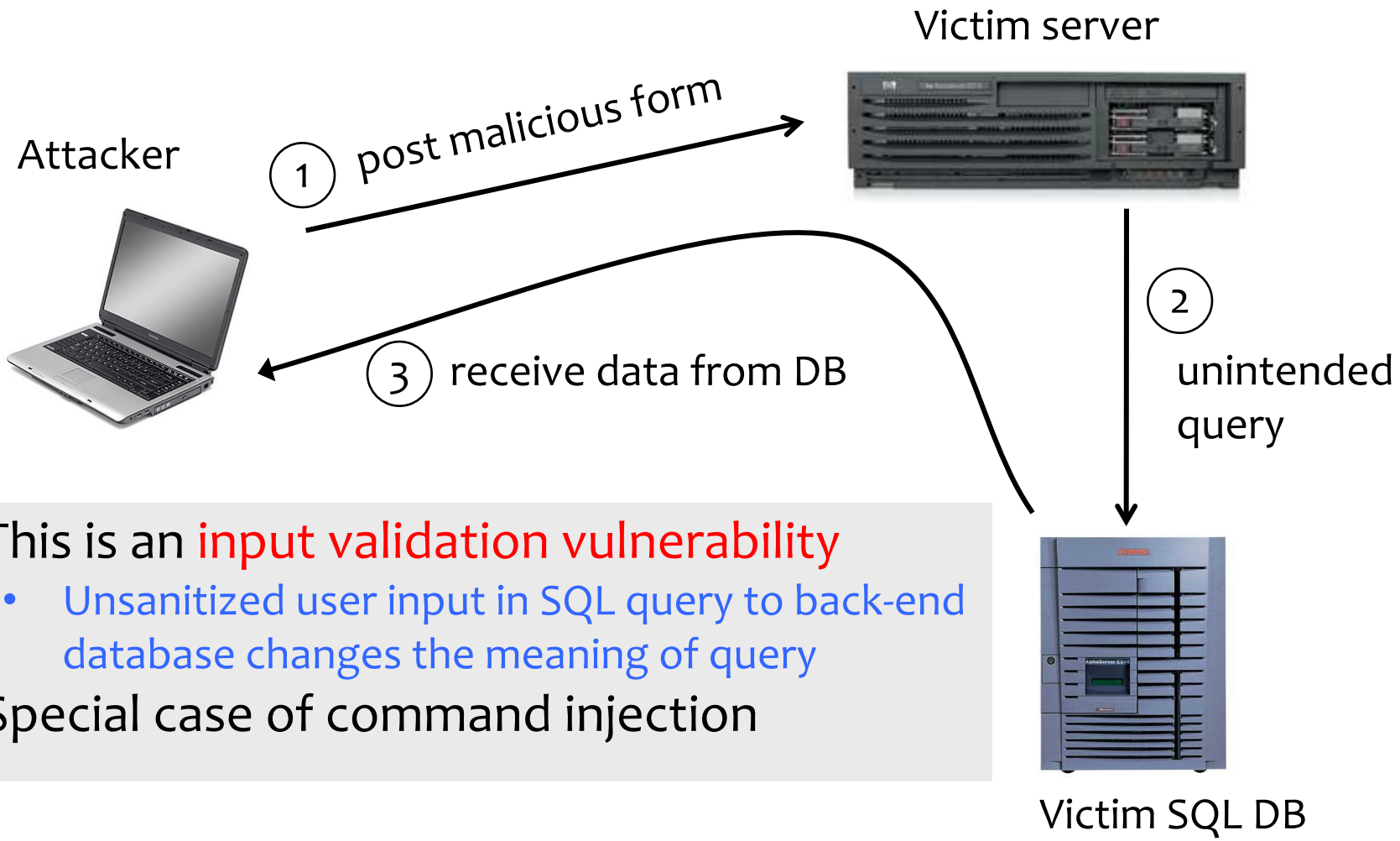


Exploits of a Mom



http://xkcd.com/327/

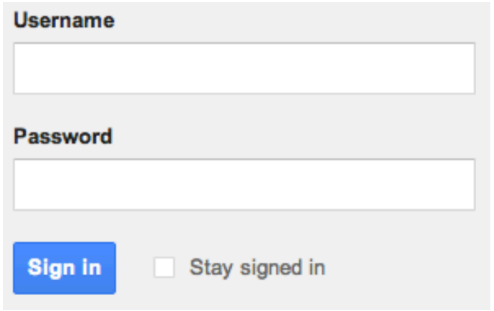
SQL Injection: Basic Idea



- This is an input validation vulnerability
  - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Special case of command injection

Authentication with Backend DB

```
set UserFound = execute(
    "SELECT * FROM UserTable WHERE
    username= ' " & form("user") & " ' AND
    password= ' " & form("pwd") & " ' ");
```



User supplies username and password, this SQL query checks if user/password combination is in the database

```
If not UserFound.EOF
    Authentication correct
  else Fail
```

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

Using SQL Injection to Log In

- User gives username **' OR 1=1 --**
- Web server executes query

```
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username= ' ' OR 1=1 -- … );
```

Always true!

Everything after -- is ignored!

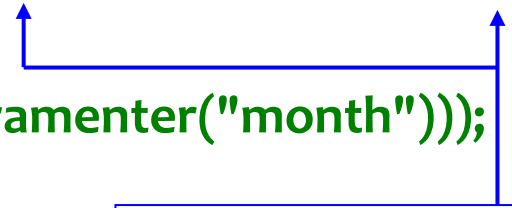
- Now all records match the query, so the result is not empty ⇒ correct "authentication"!

Preventing SQL Injection

- Validate all inputs
  - Filter out any character that has special meaning
    - Apostrophes, semicolons, percent, hyphens, underscores, …
    - Use escape characters to prevent special characters form becoming part of the query code
      - E.g.: escape(O'Connor) = O\'Connor
  - Check the data type (e.g., input must be an integer)

Prepared Statements

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
                + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

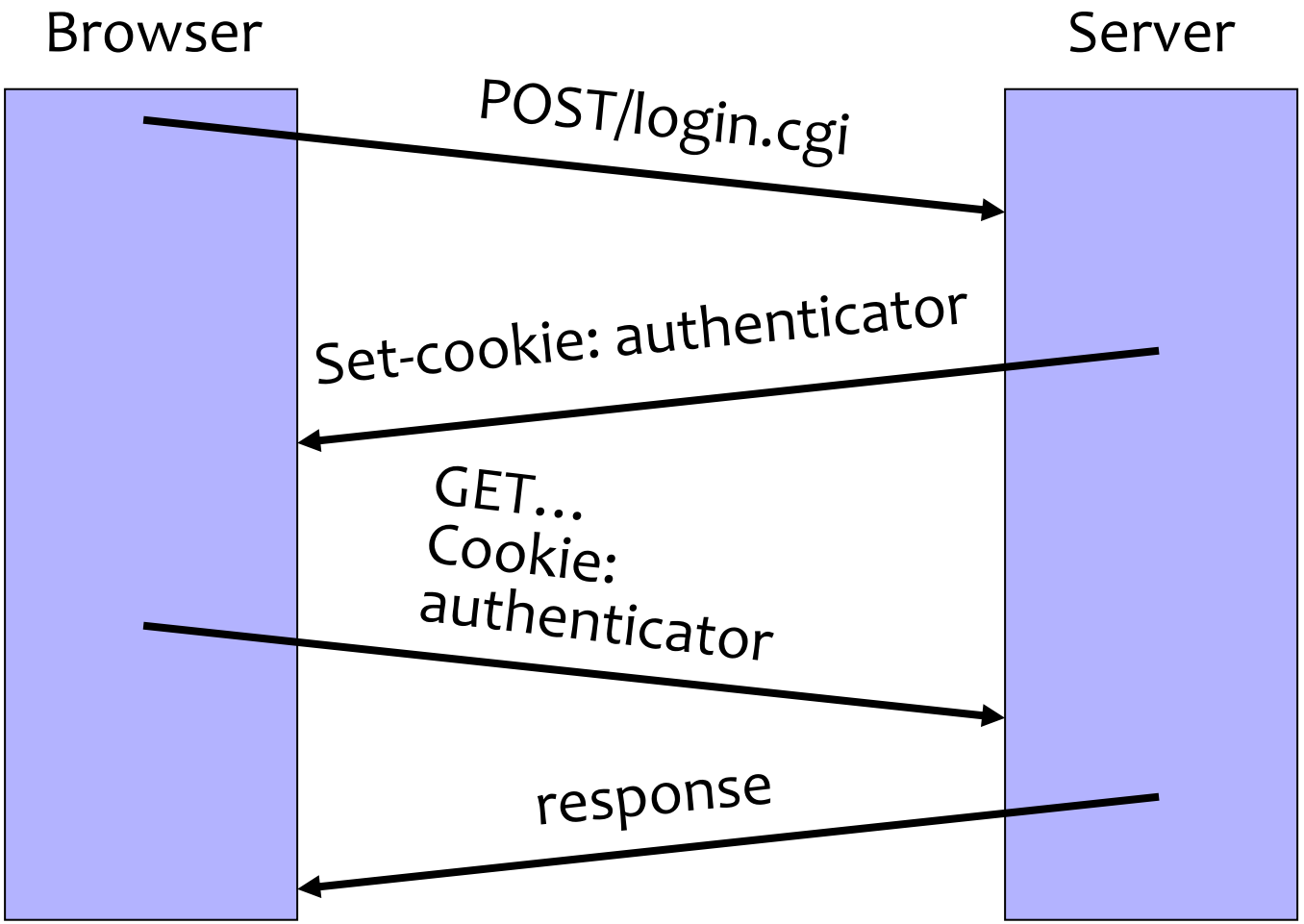


- Bind variables: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, …)

http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html

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

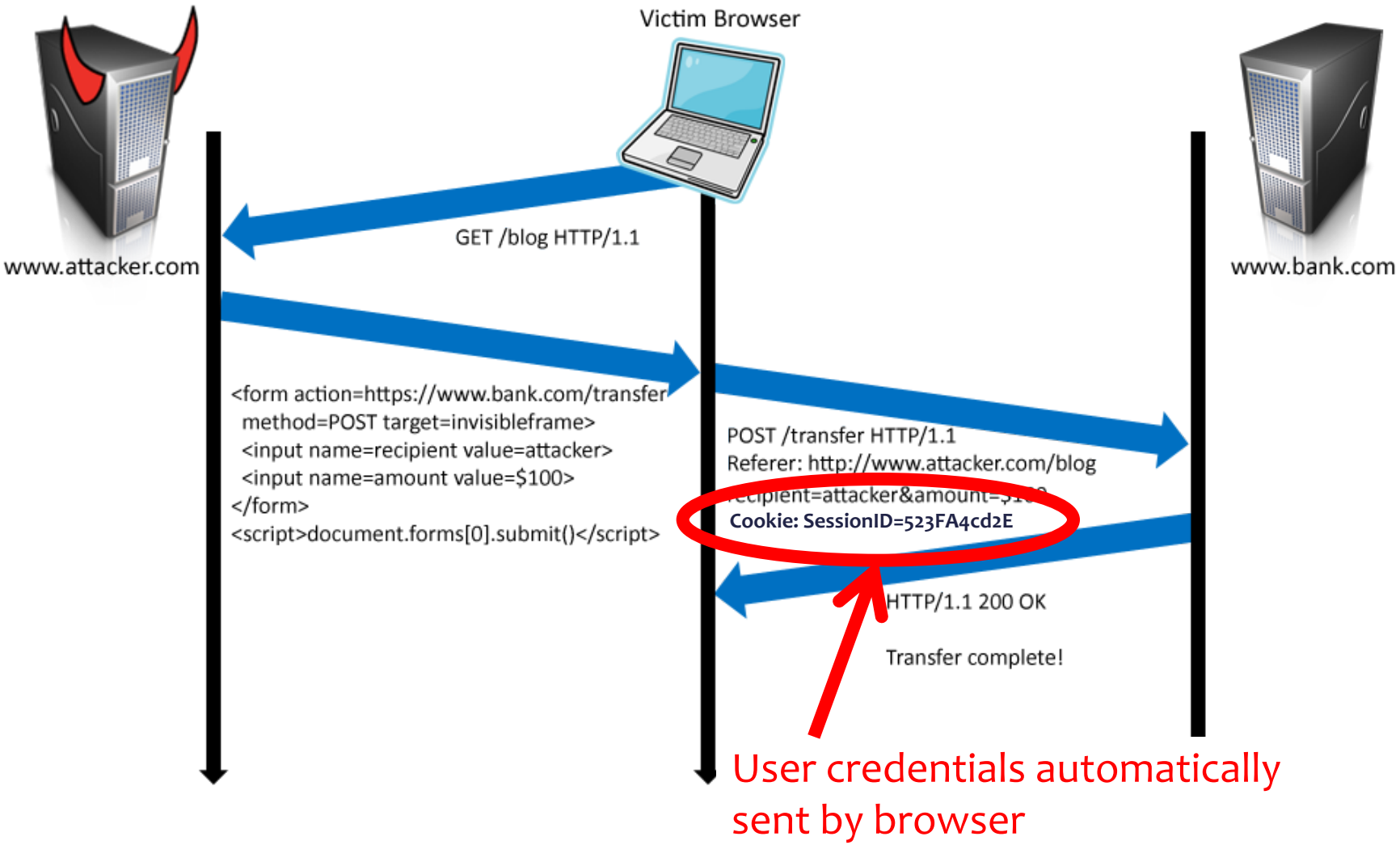
- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

```
<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



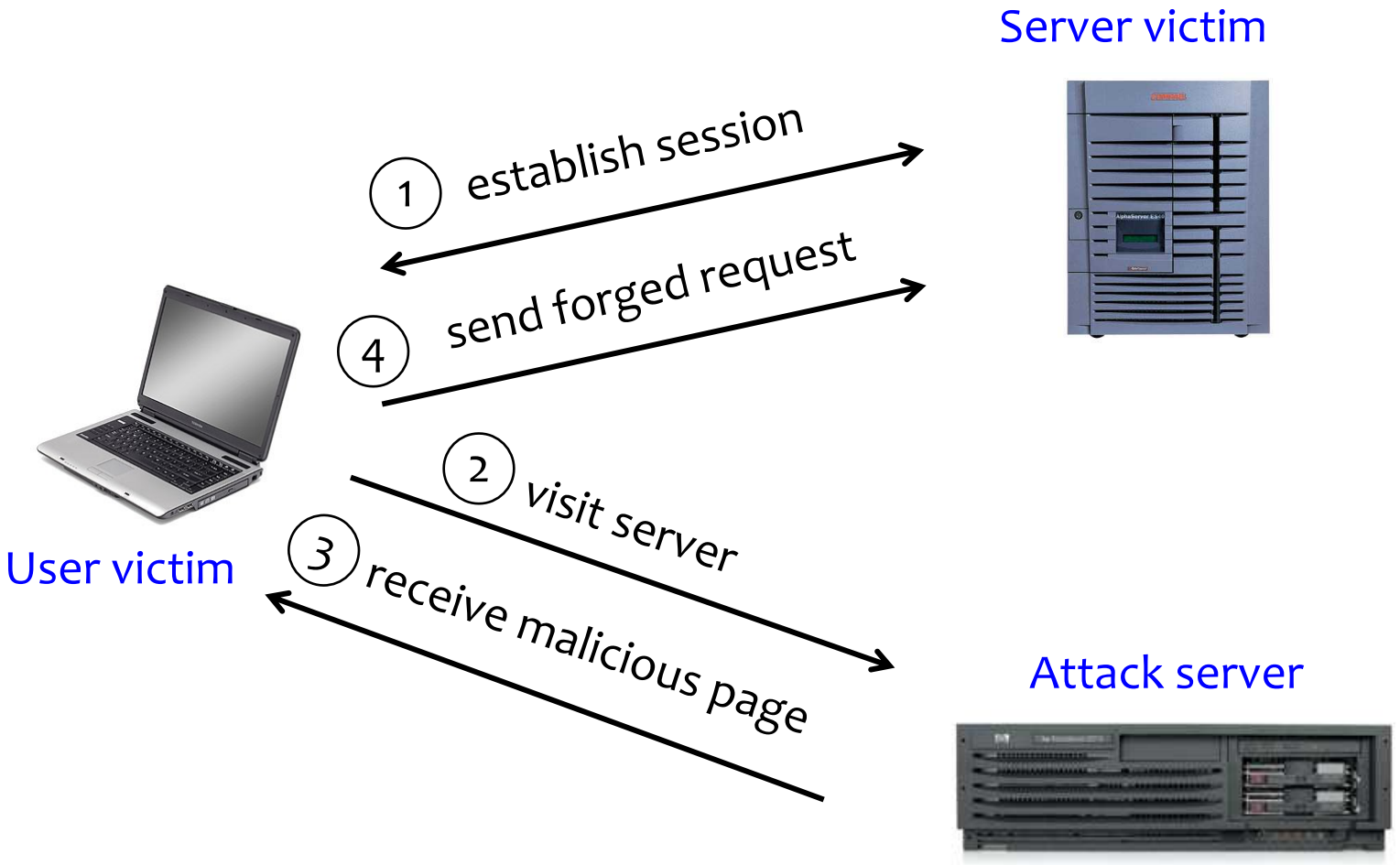
Sending a Cross-Domain POST

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

submit post

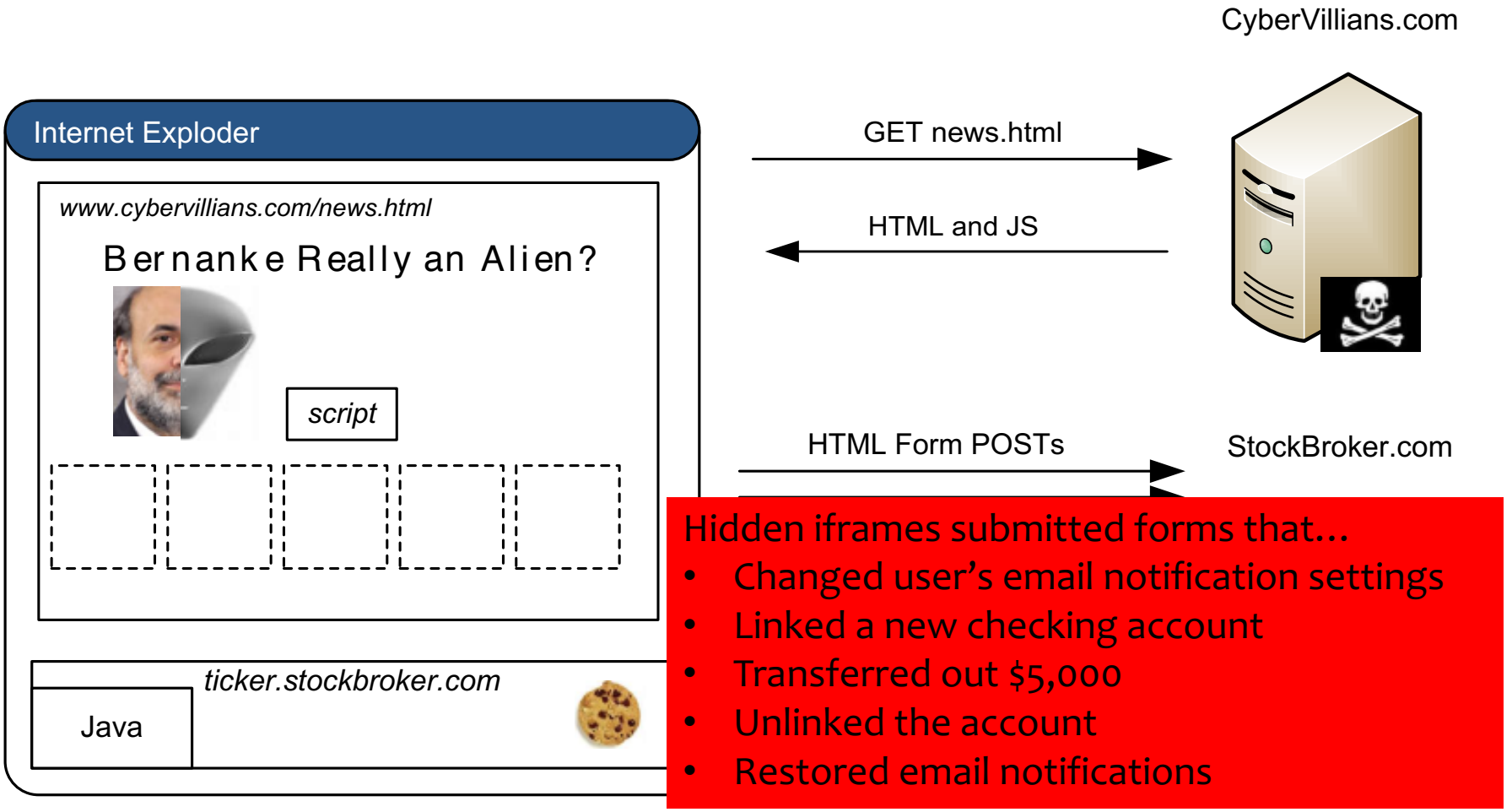
- Hidden iframe can do this in the background
- User visits a malicious page, browser submits form on behalf of the user
  - Hijack any ongoing session (if no protection)
    - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
  - Reprogram the user's home router
  -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



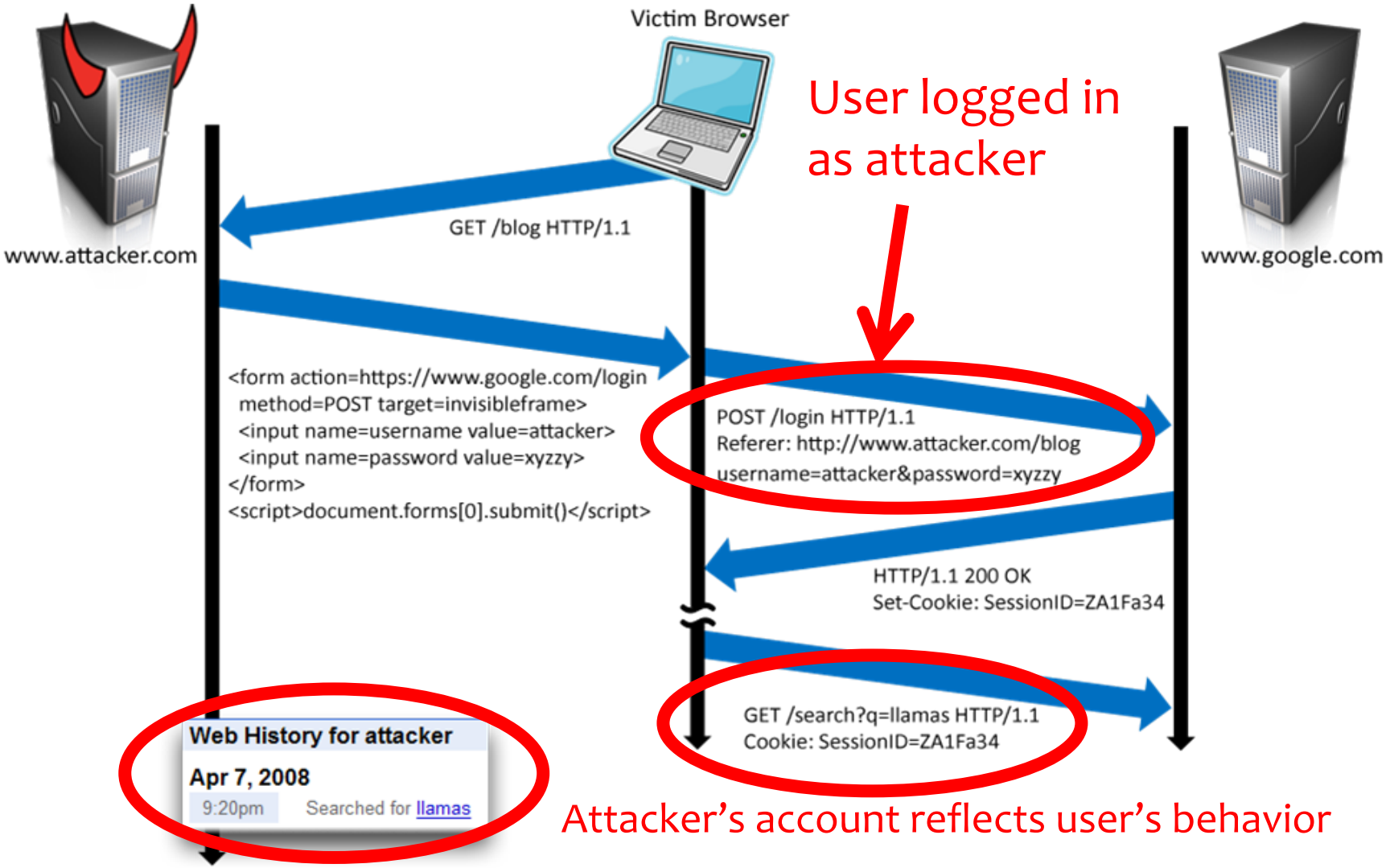
Hidden iframes submitted forms that…

- 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!



Attacker's account reflects user's behavior

XSRF Defenses

- Secret validation token


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

- Referer validation

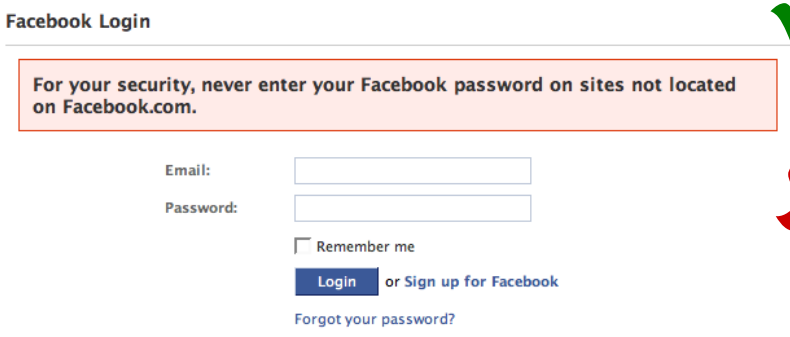

```
Referer:
http://www.facebook.com/home.php
```

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
- Referer rarely suppressed over HTTPS
  - Logins typically use HTTPS – helps against login XSRF!