CSE 484: Computer Security and Privacy

Web Security

Fall 2021

David Kohlbrenner

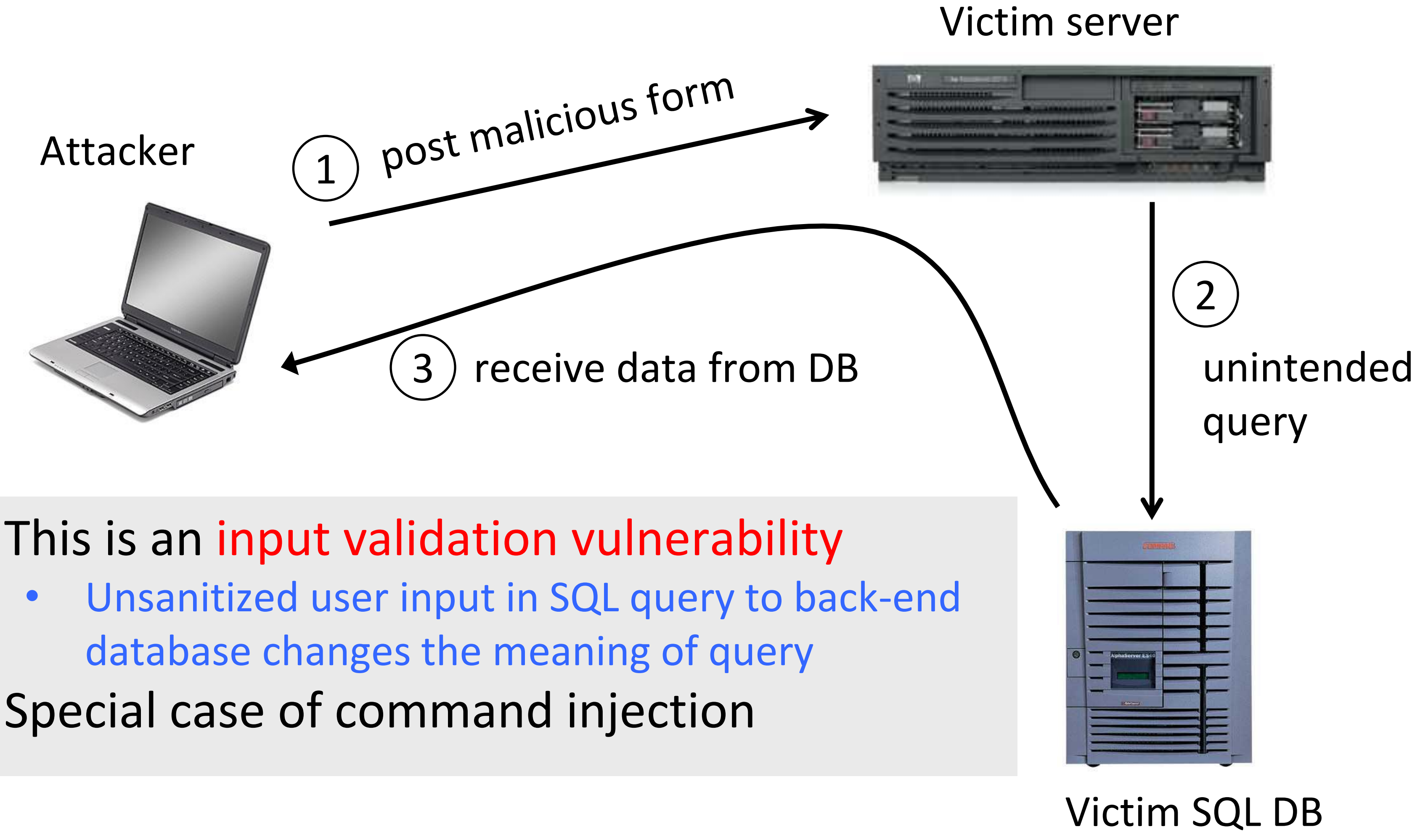
dkohlbre@cs

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

Administrivia

- Lab2 is out, due Friday 11/19
- Final Project deadline 1 Friday
  - Take a look at the syllabus for more information

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

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 <u>all</u> 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)
- Same issue as with XSS: is there anything accidentally not checked / escaped?

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

Wait, why not do that for XSS?

- “Prepared statements for HTML”?

Data-as-code

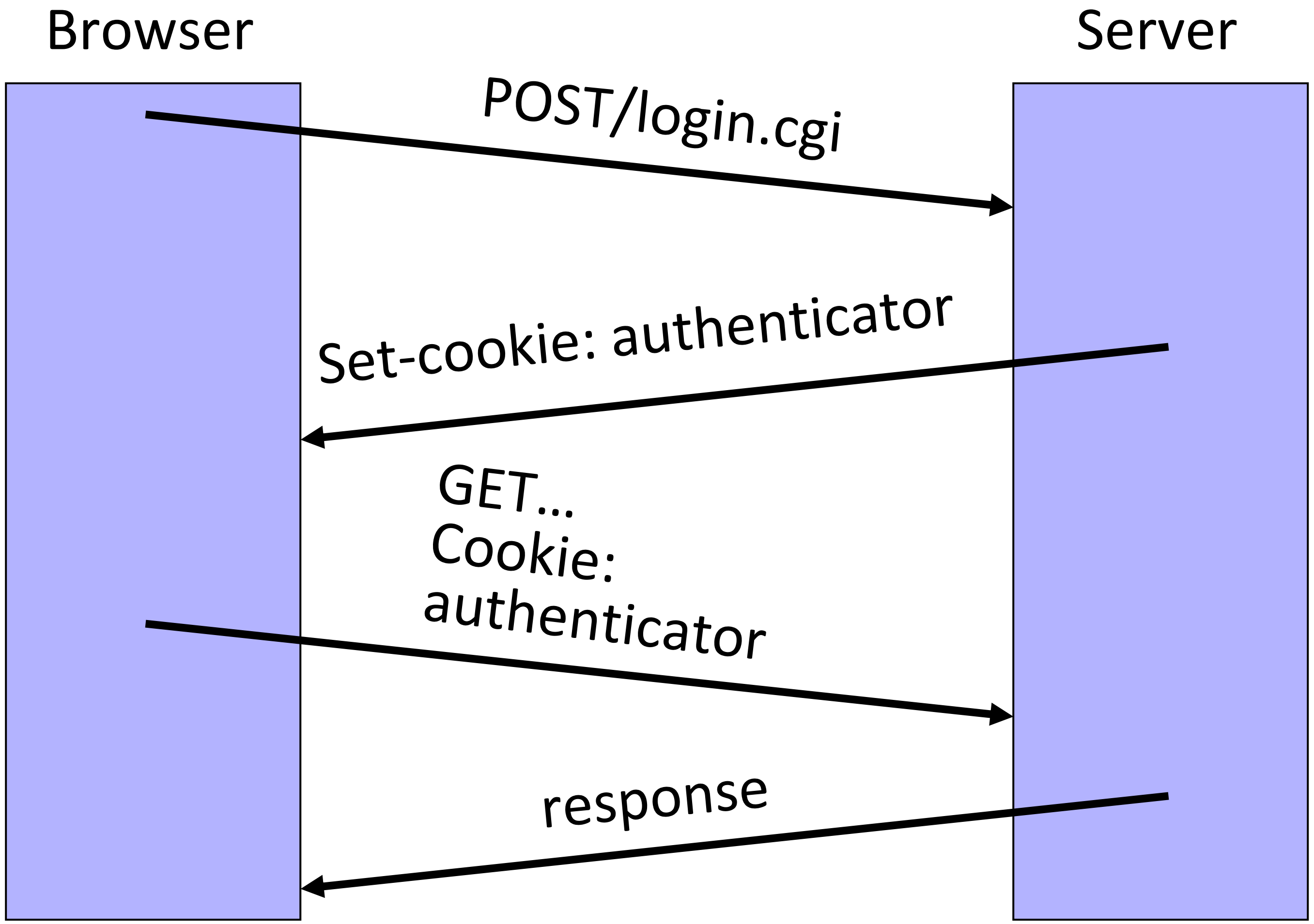
- XSS

- SQL Injection

- (Like buffer overflows)

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Review



Browser Sandbox Review

- 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

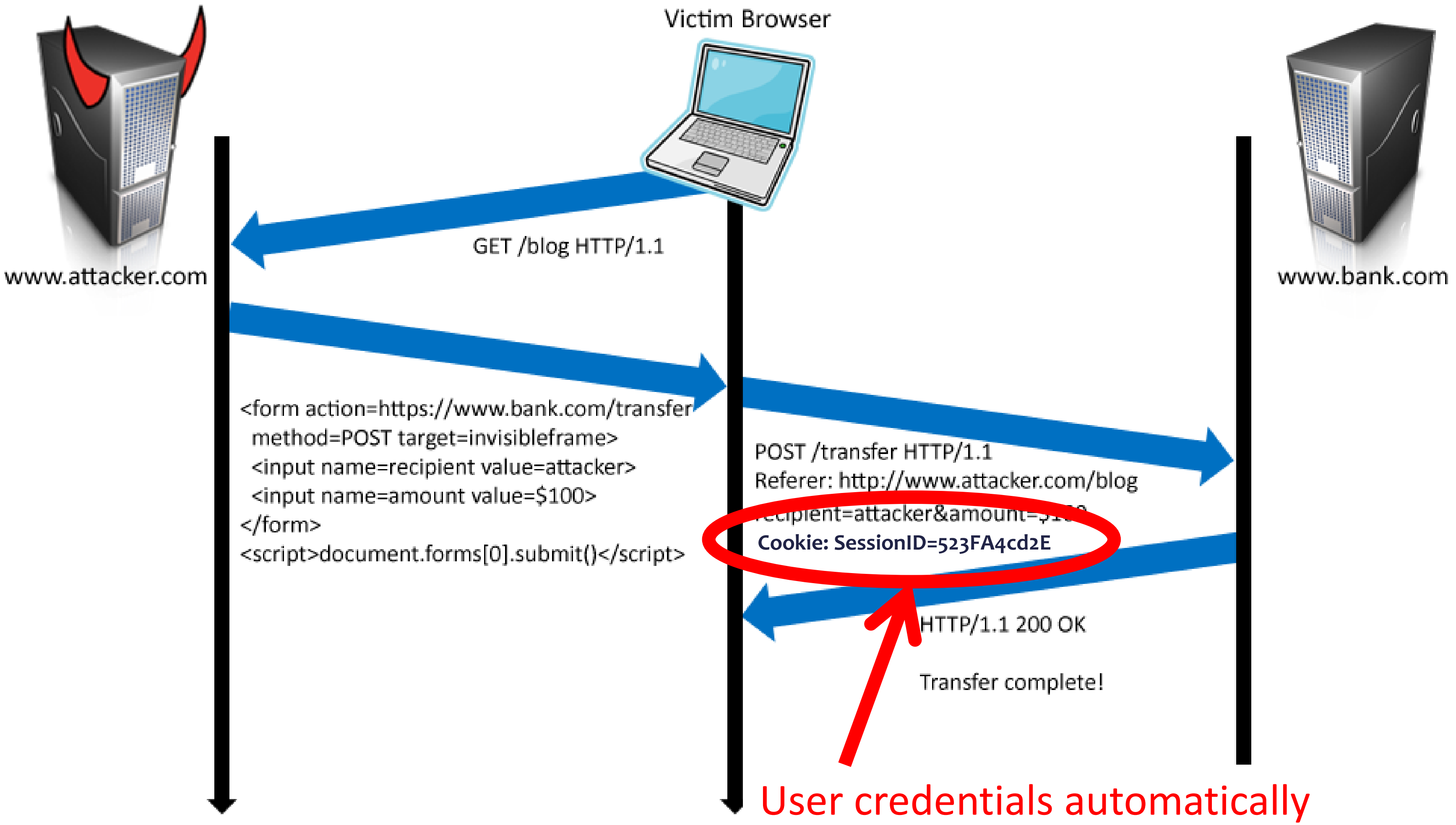
- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

```
<form  name=BillPayForm
action=http://bank.com/BillPay.php>
<input  name=recipient  value=attacker> …

<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



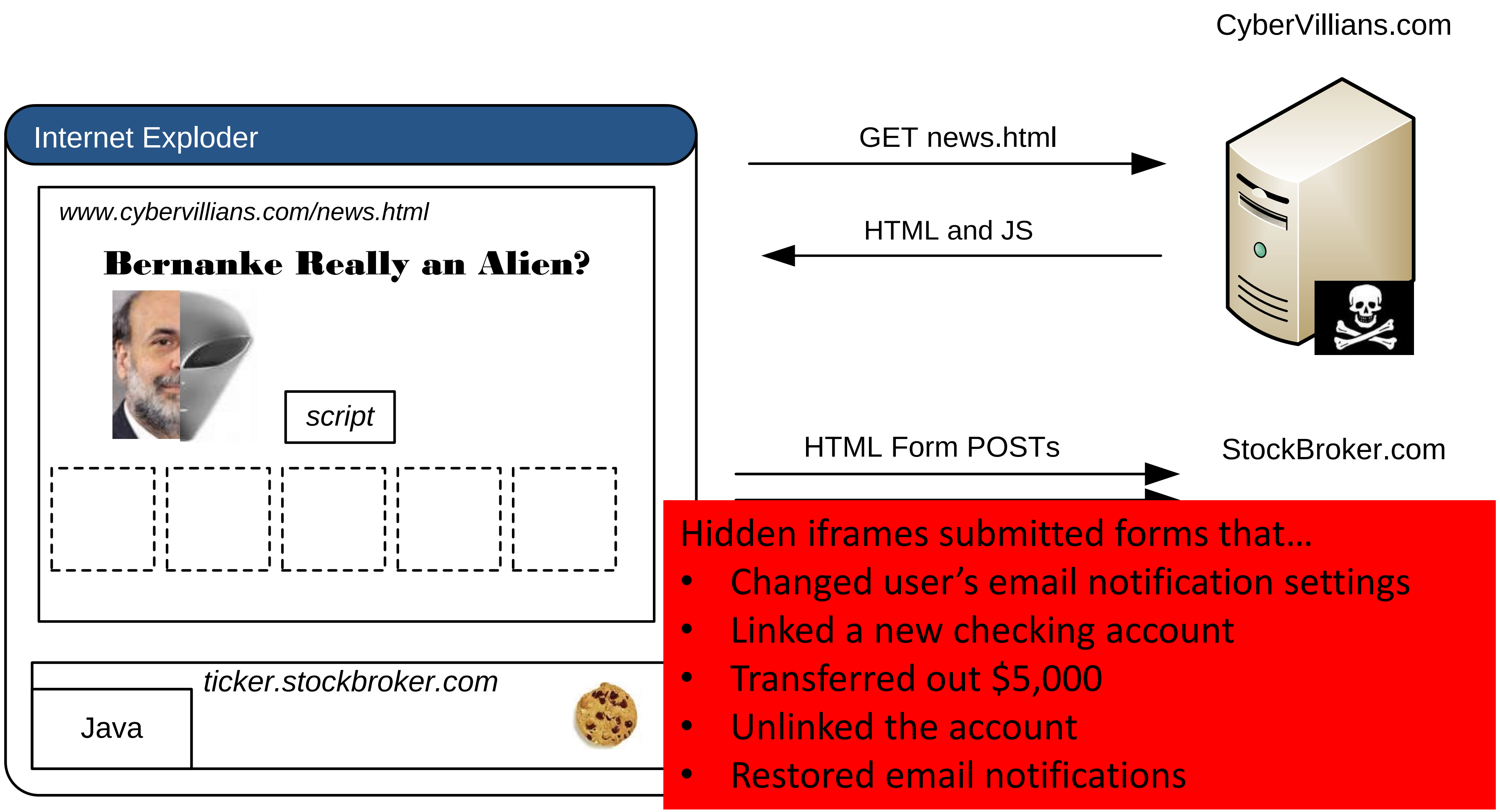
User credentials automatically sent by browser

Impact

- Hijack any ongoing session (if no protection)
  - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
- Reprogram the user's home router
- Login to the *attacker's* account
  - Why?

XSRF True Story

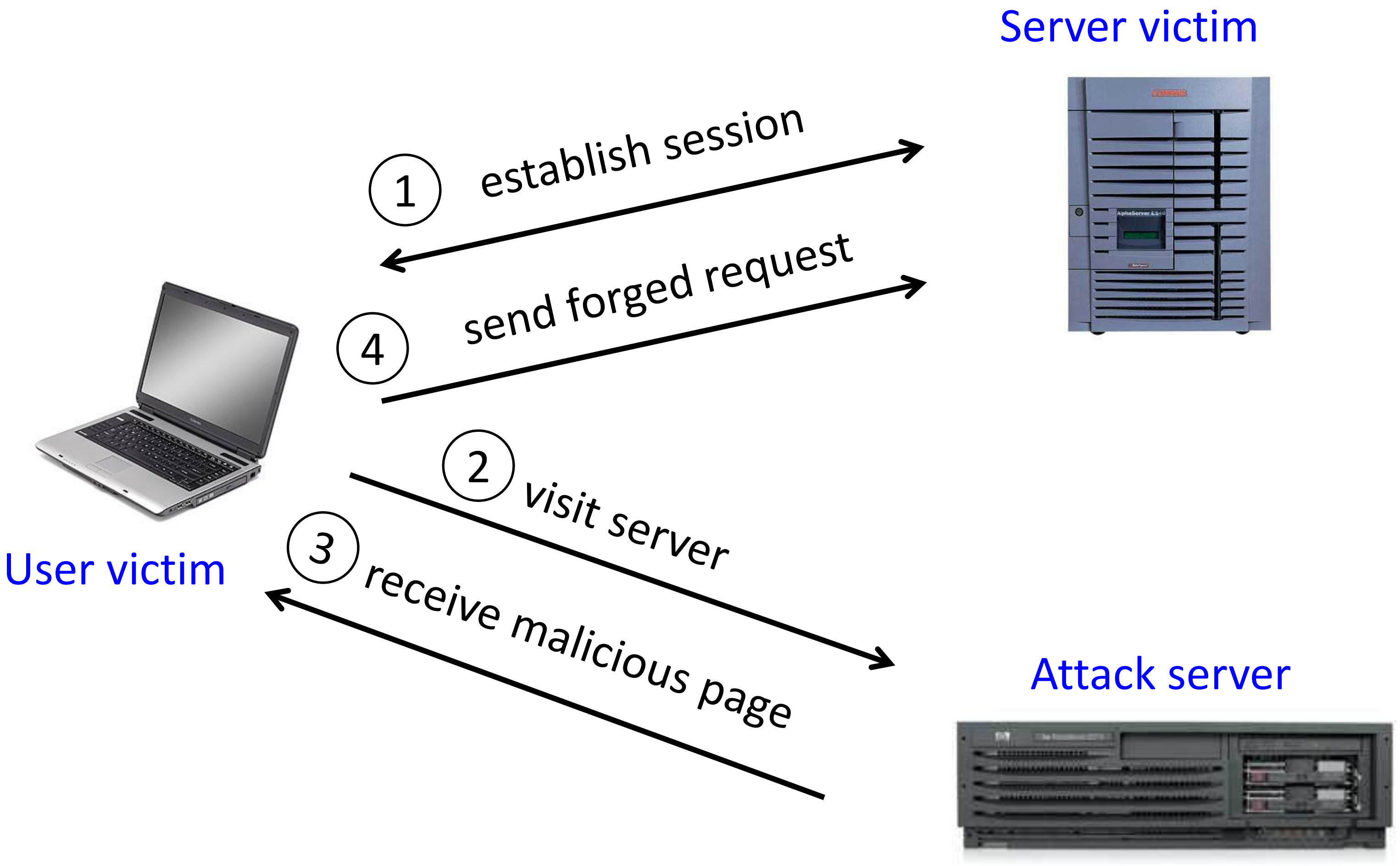
[Alex Stamos]



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

XSRF (aka CSRF): Summary



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

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)

XSRF Defenses

- Secret validation token

<input type=hidden value=23a3af01b>

- Referer validation

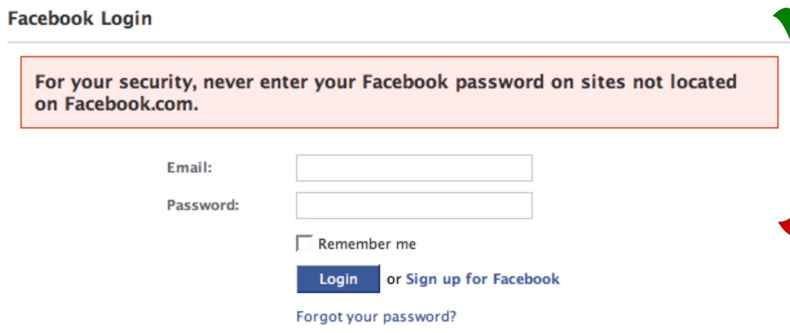
Referer:
http://www.facebook.com/home.php

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:

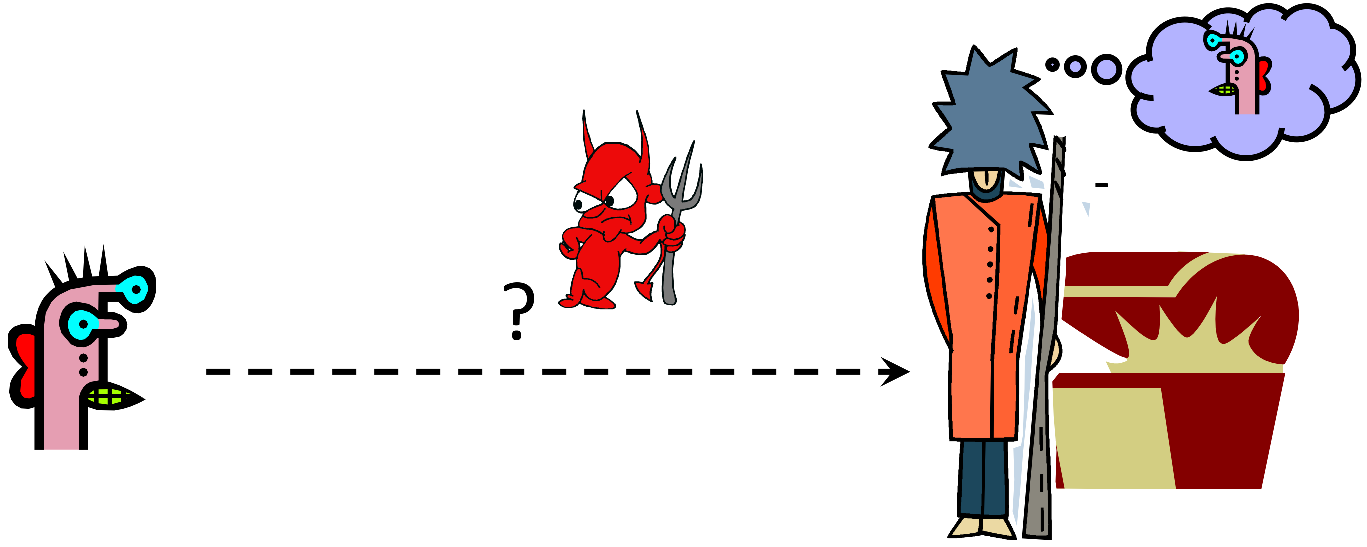
- Lenient referer checking – header is optional
- Strict referer checking – header is required

Why Not Always Strict Checking?

- Why might the referer header be suppressed?
  - Stripped by the organization's network filter
  - Stripped by the local machine
  - Stripped by the browser for HTTPS $\rightarrow$ HTTP transitions
  - User preference in browser
  - Buggy browser
- Web applications can't afford to block these users
- **Many web application frameworks include CSRF defenses today**

Authentication

Basic Problem



How do you prove to someone that you are who you claim to be?

Any system with access control must solve this problem.

Many Ways to Prove Who You Are

- What you know
  - Passwords
  - Answers to questions that only you know
- Where you are
  - IP address, geolocation
- What you are
  - Biometrics
- What you have
  - Secure tokens, mobile devices

A slightly more fundamental question

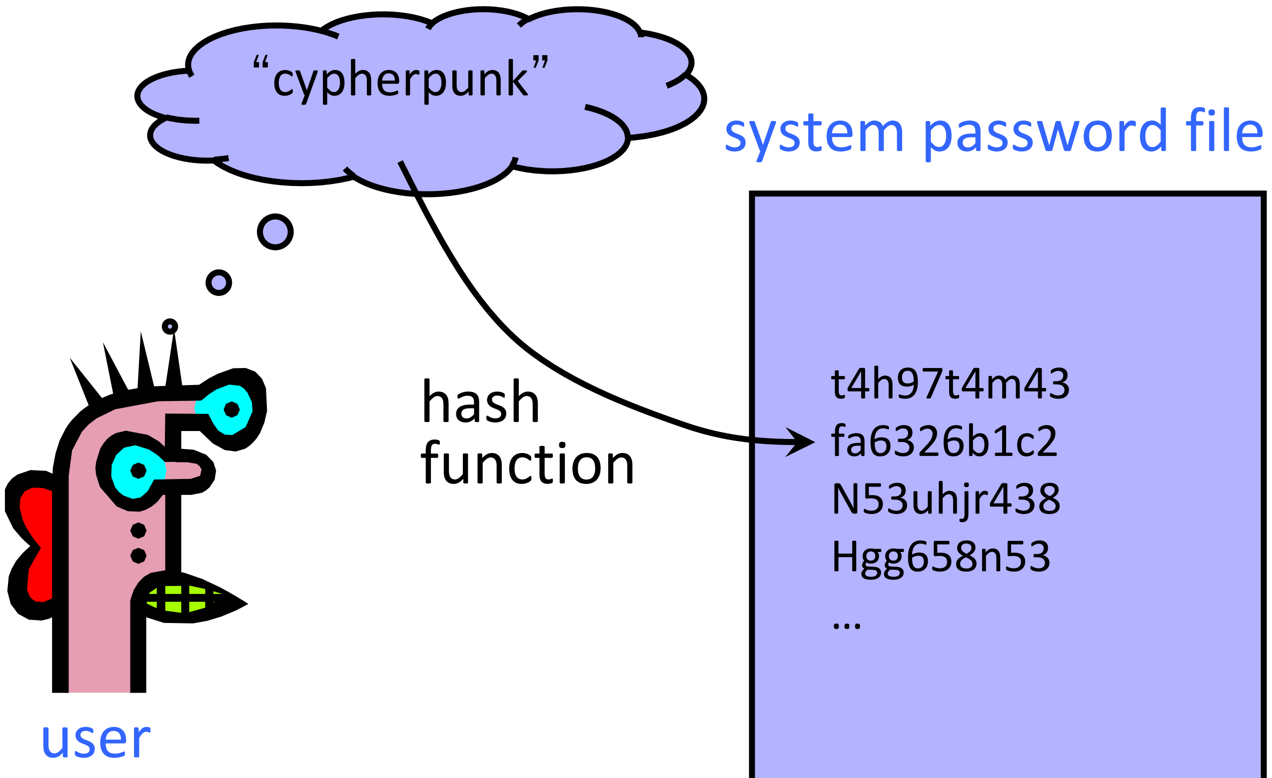
- What are we trying to prove?

Passwords and Computer Security

- In 2012, 76% of network intrusions exploited weak or stolen credentials (username/password)
  - Source: Verizon Data Breach Investigations Report
- In Mitnick's "Art of Intrusion" 8 out of 9 exploits involve password stealing and/or cracking
- First step after any successful intrusion: install sniffer or keylogger to steal more passwords
- Second step: run cracking tools on password files
  - Cracking needed because modern systems usually do not store passwords in the clear

UNIX-Style Passwords

- How should we store passwords on a server?
  - In cleartext?
  - Encrypted?
  - Hashed?



Password Hashing

- Instead of user password, store H(password)
- When user enters password, compute its hash and compare with entry in password file
  - System does not store actual passwords!
  - System itself can’t easily go from hash to password
    - Which would be possible if the passwords were encrypted
- Hash function H must have some properties
  - One-way: given H(password), hard to find password
    - No known algorithm better than trial and error
  - “Slow” to compute

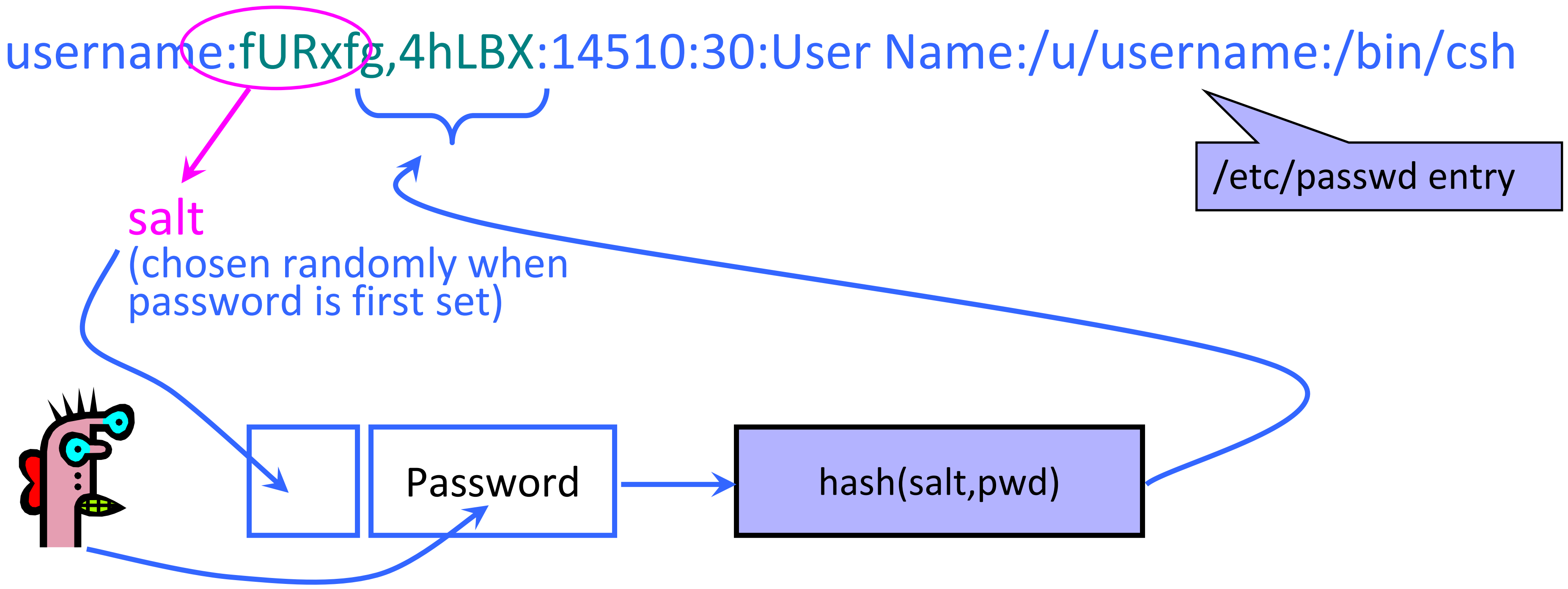
UNIX Password System

- Approach: Hash passwords
- Problem: passwords are not truly random
  - With 52 upper- and lower-case letters, 10 digits and 32 punctuation symbols, there are 94^8 == 6 quadrillion possible 8-character passwords (~2^{52})
  - **BUT:** Humans like to use dictionary words, human and pet names == 1 million common passwords

Dictionary Attack

- Dictionary attack is possible because many passwords come from a small dictionary
  - Attacker can pre-compute H(word) for every word in the dictionary – this only needs to be done once!
    - This is an offline attack
    - Once password file is obtained, cracking is instantaneous
  - Sophisticated password guessing tools are available
    - Take into account freq. of letters, password patterns, etc.

Salt



- Users with the same password have <u>different</u> entries in the password file
- Offline dictionary attack becomes much harder

Advantages of Salting

- Without salt, attacker can pre-compute hashes of all dictionary words once for all password entries
  - Same hash function on all UNIX machines
  - Identical passwords hash to identical values; one table of hash values can be used for all password files
- With salt, attacker must compute hashes of all dictionary words once for each password entry
  - With 12-bit random salt, same password can hash to 2^{12} different hash values
  - Attacker must try all dictionary words **for each salt value** in the password file
- Pepper: Secret salt (not stored in password file)

Shadow Password

username:x:14510:30:User Name:/u/username:/bin/csh

/etc/passwd entry

Hashed password is no longer stored in a world-readable file

Hashed passwords are stored in /etc/shadow file which is only readable by system administrator (root)