CSE 484 / CSE M 584: Computer Security and Privacy

Web Security

Autumn 2018

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

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

Admin

- HW2: Due Nov 7, 4:30pm

- Looking ahead, rough plan:
- Lab 2 out ~Nov 5, due ~Nov 19 (Quiz Section on Nov 8)
- HW 3 out ~Nov 19, due ~Nov 30
- Lab 3 out ~Nov 26, due Dec 7 (Quiz Section on Nov 29)

Admin

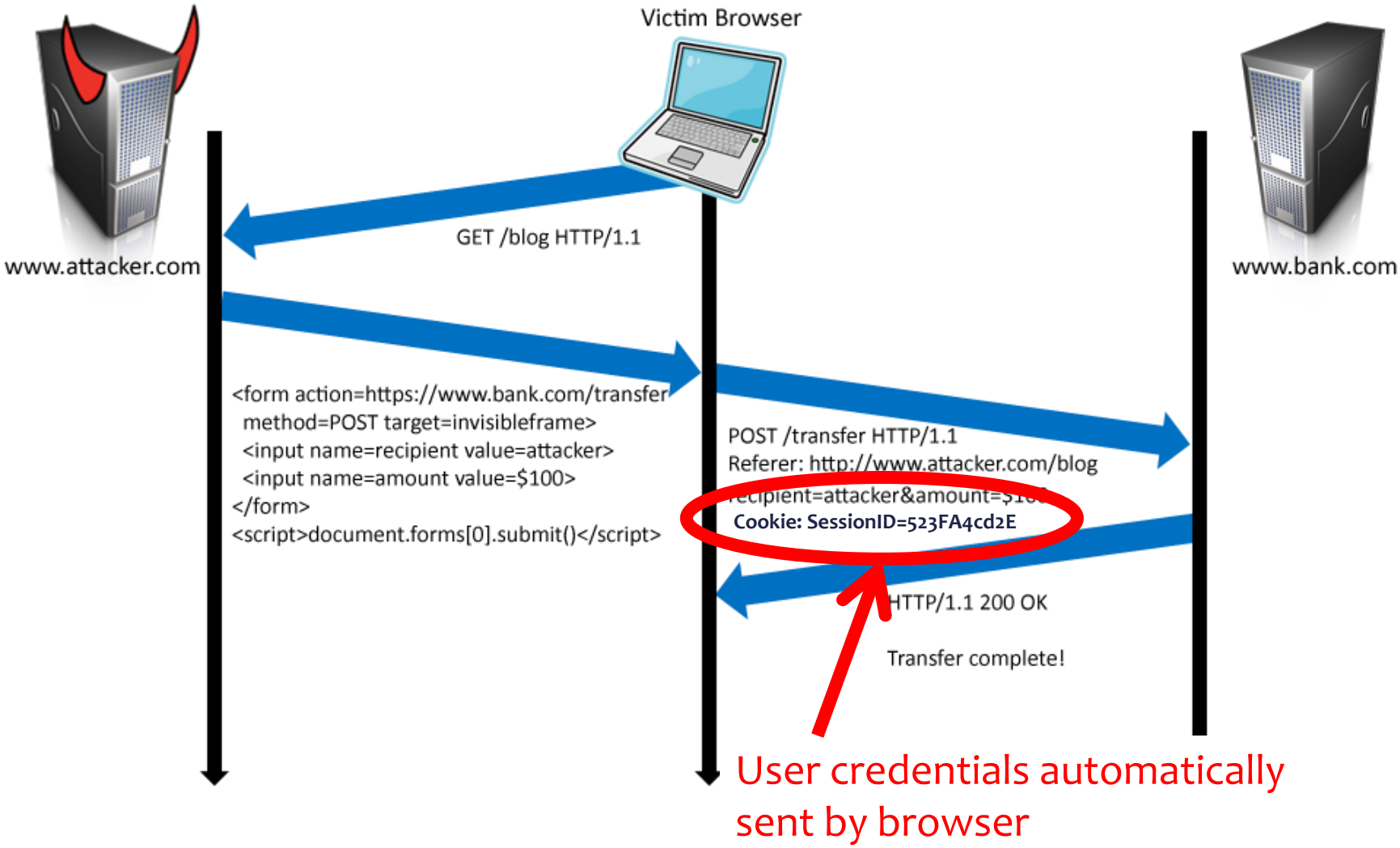
- Final Project Proposals: Nov 16 – group member names and brief description
- Final Project Checkpoint: Nov 30 – preliminary outline and references
- Final Project Presentation: Dec 10 – 12-15-minute video – **must** be on time
- Explore something of interest to you, that could hopefully benefit you or your career in some way – technical topics, current events, etc

Next Week

- Monday (Nov 5): Lecture on Lab 2
- Monday (Nov 5): No 11:30am office hours for me; TA office hours are happening
- Thursday (Nov 8): Quiz Sections -> Extended Lab 2 Office Hours

Cross-Site Request Forgery (CSRF/XSRF)

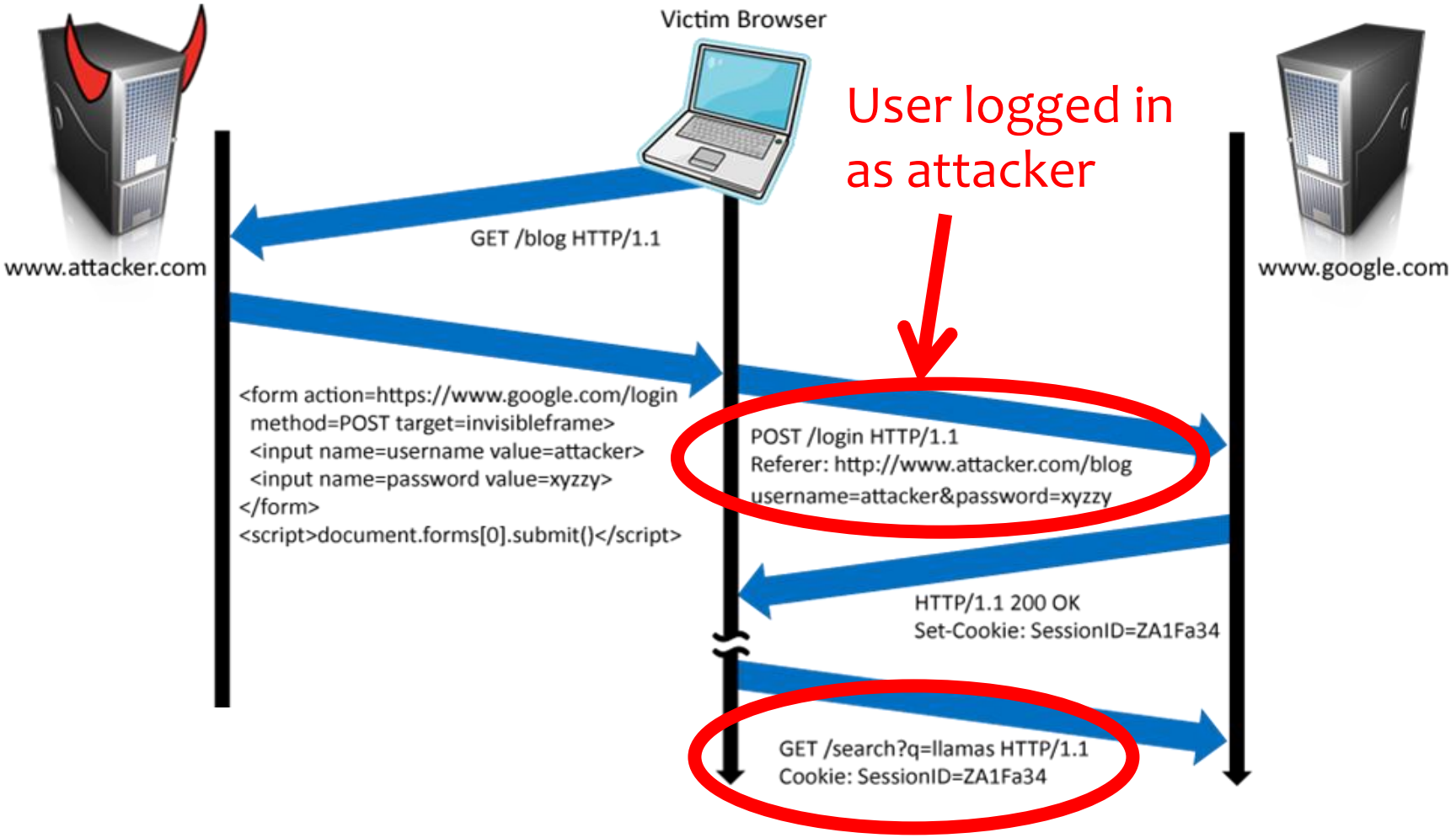
Cookies in Forged Requests



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

Login XSRF: Attacker logs you in as them!



Broader View of CSRF

- 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

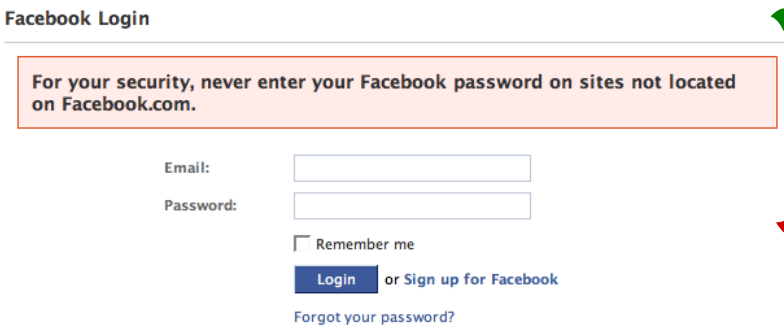

```
Referer:
http://www.facebook.com/home.php
```

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 → 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

Injection

Injection Attacks

- http://victim.com/copy.php?name=username
- copy.php includes

  system("cp temp.dat $name.dat")
- User calls

  http://victim.com/copy.php?name="a; rm *"
- copy.php executes

  system("cp temp.dat a; rm *.dat");

Basic Issues

- User-supplied data is not validated, filtered, or sanitized by application
- User input directly used or concatenated to a string that is used by an interpreter

- Common Injections: SQL, NoSQL, Object Relational Mapping (ORM), LDAP, Object Graph Navigation Library, …

More Examples

- SQL application uses untrusted data in this SQL call

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

- Also, be careful with frameworks, e.g., Hibernate Query Language (HQL) call

```
Query HQLQuery = session.createQuery("FROM
accounts WHERE custID='" +
request.getParameter("id") + "'");
```

- Attacker sets id to ' or '1'='1

```
http://example.com/app/accountView?id=' or '1'='1
```

- Result in both cases: return all records in database

Defenses

- Use safe APIs, e.g., prepared statements in SQL with parameterized queries
  - Define all the SQL code, then pass in each parameter
  - Separates code from data
- Whitelist-based server-side input validation
- Escape special characters
- Use LIMIT (and other) SQL controls within queries to prevent mass disclosure of records

- Remember Defense in Depth, Least Privilege, etc.

XML External Entities

XML External Entities

- Consider a web application that accepts XML input, parses it, and outputs the result (or includes untrusted input in XML documents)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar "World">
]>
<foo>
  Hello &bar;
</foo>
```

- Parses as

  Hello World

But What About

- Consider an attacker uploading this XML document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

- Attacker attempting to extract information from server

But What About

- Consider an attacker uploading this XML document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
<foo>&xxe;</foo>
```

- Attacker attempting to probe a private network

But What About

- Consider an attacker uploading this XML document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
<foo>&xxe;</foo>
```

- Attacker attempting a DoS by including a potentially never-ending file

Why Call “Server Side Request Forgery?”

What to Do?

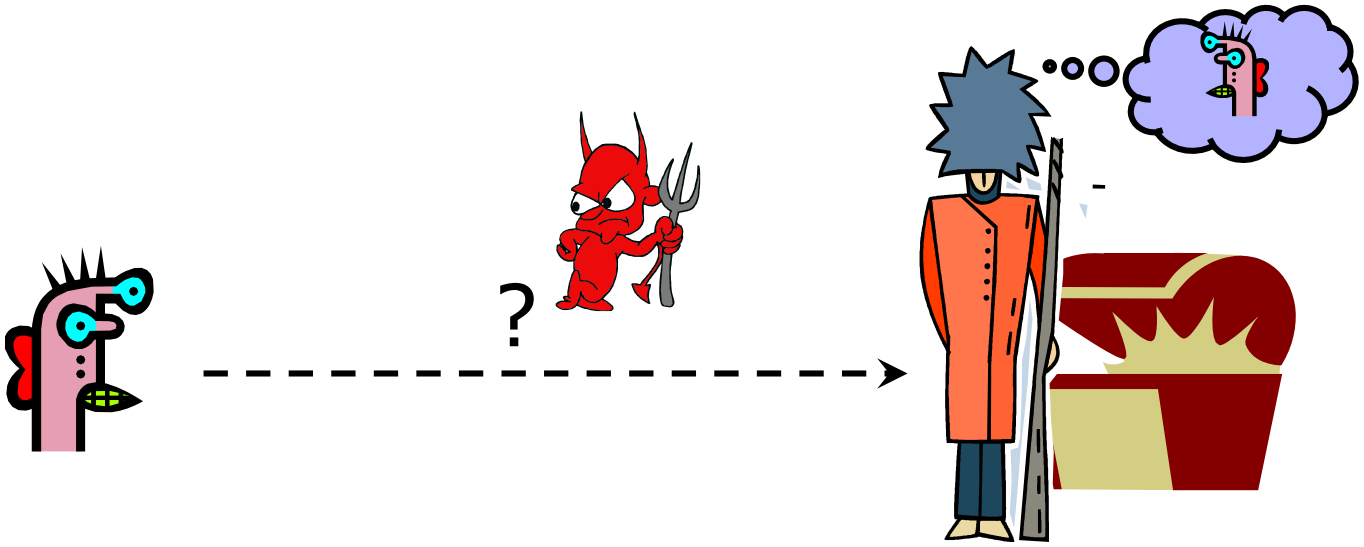
- Use less complex data formats, such as JSON
- Disable XML external entities and DTD processing in all XML parses
- Whitelist-based server-side input validation

- OWASP very useful source here as well

Authentication

Another “Ten Most Critical Web Application Security Risks”

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

Passwords and Computer Security

- In 2012, 76% of network intrusions exploited weak or stolen credentials (username/password)
  - Source: Verizon Data Breach Investigations Report
- 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 (how are they stored?)
- In Mitnick's "Art of Intrusion" 8 out of 9 exploits involve password stealing and/or cracking

Password Storage

- Recall discussions from crypto section
  - Don't store plaintext passwords
  - Don't use encrypted passwords
  - Use hashed passwords
  - Hash a salt along with the password, and store the salt and the hashed salt+password on the server

Other Password Security Issues

- Keystroke loggers
  - Hardware
  - Software (spyware)
- Shoulder surfing
- Same password at multiple sites
- Broken implementations
  - TENEX timing attack

Examples from One Company

Even More Issues

- Usability
  - Hard-to-remember passwords?
  - Carry a physical object all the time?
- Denial of service
  - Attacker tries to authenticate as you, account locked after three failures
- Social engineering