Admin

• **Lab 2**
  – Granting access starting today
  – Please sign up if you haven’t already

• **Final project**
  – First checkpoint deadline Friday

• **This week**
  – No class or office hours Wednesday (Veterans’ Day)
  – Guest lecture Friday (Charlie Reis, Google, web security)
SQL Injection
Typical Login Prompt
Typical Query Generation Code

```php
$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

Web browser (Client) \[\xrightarrow{\text{Enter Username \\ & Password}}\] Web server \[\xrightarrow{\text{SELECT passwd FROM USERS WHERE uname IS ‘\$user’}}\] DB
Normal Login

Web browser (Client) → Enter Username & Password → Web server → SELECT passwd FROM USERS WHERE uname IS 'franzi' → DB
Malicious User Input

![Image of a login page with malicious input]

- **Enter User Name:** '; DROP TABLE USERS; --
- **Enter Password:** 

This image demonstrates an example of malicious user input, where the user name field is being exploited to execute a SQL injection attack.
SQL Injection Attack

Web browser (Client) → Web server → DB

Enter Username & Password

SELECT passwd FROM USERS WHERE uname IS '' ; DROP TABLE USERS; -- ’

Eliminates all user accounts
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
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 from 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.getParameter("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, ...)

Cross-Site Request Forgery (CSRF/XSRF)
Cookie-Based Authentication Redux

Browser

POST/login.cgi

Set-cookie: authenticator

GET...

Cookie: authenticator

response

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

  ```javascript
  <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]

Internet Exploder

Bernanke Really an Alien?

script

www.cybervillians.com/news.html

ticker.stockbroker.com

Java

GET news.html

HTML and JS

HTML Form POSTs

StockBroker.com

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

CyberVillians.com
XSRF (aka CSRF): Summary

1. establish session
2. visit server
3. receive malicious page
4. send forged request

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

  ![Rails Logo]  ![Facebook Logo]

  `<input type=hidden value=23a3af01b>

• Referer validation

  Referer:
  http://www.facebook.com/home.php
Add Secret Token to Forms

- "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

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

Referer:
- ?

Facebook Login

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

Email:  
Password:

Remember me
Login or Sign up for Facebook
Forgot your password?
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