How to Build a Secure Login

Ben Broussard
Kedalion Security

Contents

- How Authentication works
  - Pre-Login
  - Login Page
  - Login Redirect
  - Logged In
  - Log Out
- Attacks and defenses on each step

Pre-Login

- **Pre-Login**
- Login Page
- Login Redirect
- Logged In
- Log Out

- Users get to the site in many ways: Search engine, Bookmarks, Links from emails, Direct URL entry, iframes from other sites.
- Request/Response model.
- Users shouldn't be able to complete most actions before logging in, but they may be able to begin actions such as adding items to a cart or setting up a session.
- Account Creation
- Password Reset

REQUEST

GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; U; Lin...
Accept: text/html,application/xhtml+xml,applica...
Keep-Alive: 115

RESPONSE

HTTP/1.1 200 OK
Date: Fri, 29 Apr 2011 17:12:13 GMT
Set-Cookie: skin=noskin; path=/; domain=.example.com; expires=Fri, 29-Apr-2011 17:12:13 GMT
Content-Type: text/html; charset=ISO-8859-1
Set-cookie: session-id=176-9381406-6210335; path=/; domain=.example.com; expires=Tue Jan 01 08:00:01 2036 GMT
Content-Length: 156046

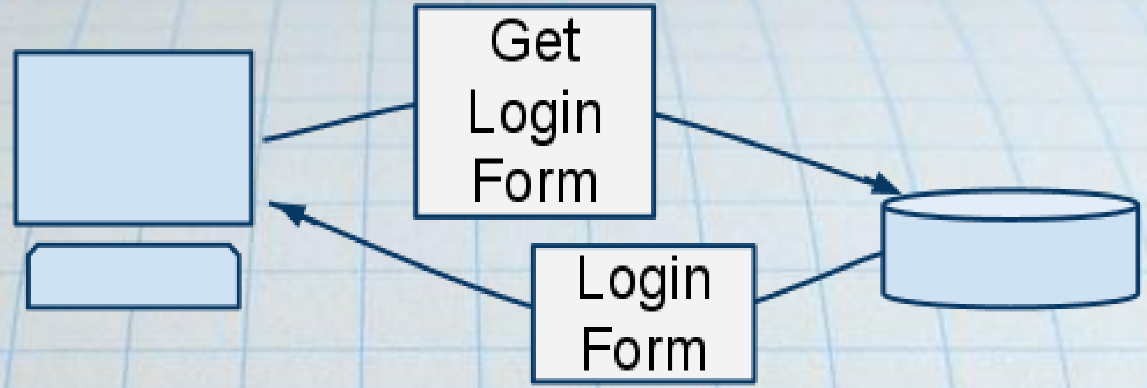
<html>
... **web page**

Login Page

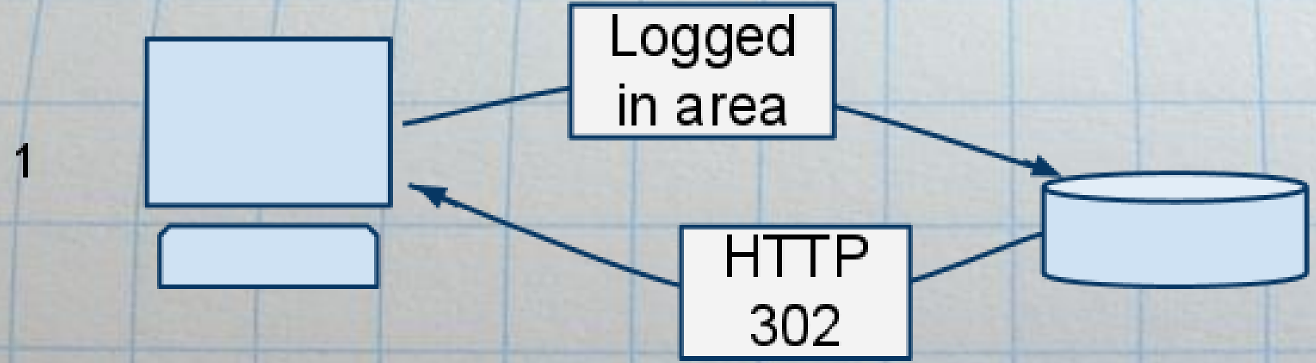
- Pre-Login
- **Login Page**
- Login Redirect
- Logged In
- Log Out

- Users can get to the login page by:
  - Clicking on the login link on the site or from an email or another site.
  - Attempting to go to a logged in page without being logged in.
  - Making a request to a logged in page after the session has expired.
- The login page needs to know where to send the user after successful login.
- Input can include a username, password, pre-login cookie, anti-CSRF token, CAPTCHA, and even a second factor such as an RSA token.

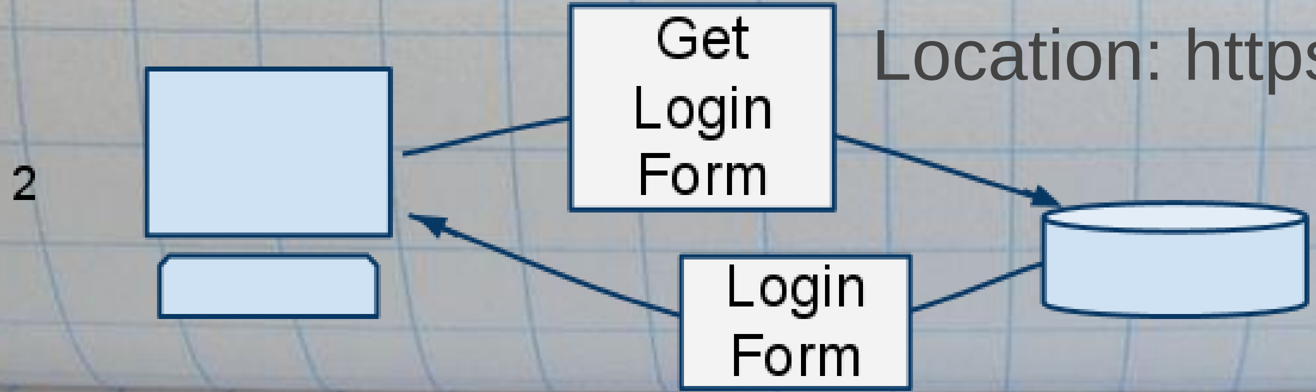
Clicked on Login link



Redirected to Login



HTTP/1.1 302 Found
Location: https://example.com/login/



1. Request to Logged in Page:

GET mail/inbox.php?email_id=11&action=mark_as_read HTTP/1.1

2. 302 Response containing

Set-cookie: go_to=/mail/inbox.php?email_id=11&action=mark_as_read
Location: https://example.com/login.php

3. Request to https://example.com/login.php

4. Response containing Login page:

HTTP/1.1 200 OK
... Other Headers

<html>
... **Login Form**

5. Request containing credentials:

POST /login.php HTTP/1.1
Host: example.com
Cookie: anonymous_session_id=ff5f109f765de12d3a83ce578e9d44ef; go_to=/mail/inbox.php?email_id=11&action=mark_as_read

username=ben&password=myrealpassword&csrf_token=6108d48838...

Login Redirect

- Pre-Login
- Login Page
- **Login Redirect**
- Logged In
- Log Out

- Upon successful verification of the user's credentials, a redirection response which contains a Set-Cookie header is returned.
  - Usually an HTTP 302 Found response with a Location header.
  - Sometimes a webpage is returned which includes a javascript or meta tag redirect.
- This new cookie is the logged in session cookie.

1. Response from successful login:

HTTP/1.1 302 Found
Set-Cookie: session_id=617372ea63040f780b16dd992122e170; path=/; secure; HttpOnly
Location: https://example.com/mail/inbox.php?email_id=11&action=mark_as_read

2. Request to Location value:

GET /mail/inbox.php?email_id=11&action=mark_as_read HTTP/1.1
Host: example.com
Cookie: session_id=617372ea63040f780b16dd992122e170

3. Response containing logged in page:

HTTP/1.1 200 OK
... Other Headers

<html>
... **Logged in Page**

Logged In

- Pre-Login
- Login Page
- Login Redirect
- **Logged In**
- Log Out

- Now that the user is logged in, they can take sensitive actions and look at sensitive data.
- The user stays logged in because the browser adds the Cookie header to every request (with the appropriate domain, path, flags, etc.).
- Often users have to fill out long forms that take longer than the inactivity logout period.
- Users may have multiple tabs open which makes it difficult to impose an order on their actions.

REQUEST

POST /payroll/directdeposit.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.16) Gecko/20110323 Ubuntu/10.04 (lucid) Firefox/3.6.16
Cookie: session_id=617372ea63040f780b16dd992122e170

routing_nbr=111111111&acct_nbr=123412341234&csrf_token=c1446f6da166450281c91108551ae9b6

RESPONSE

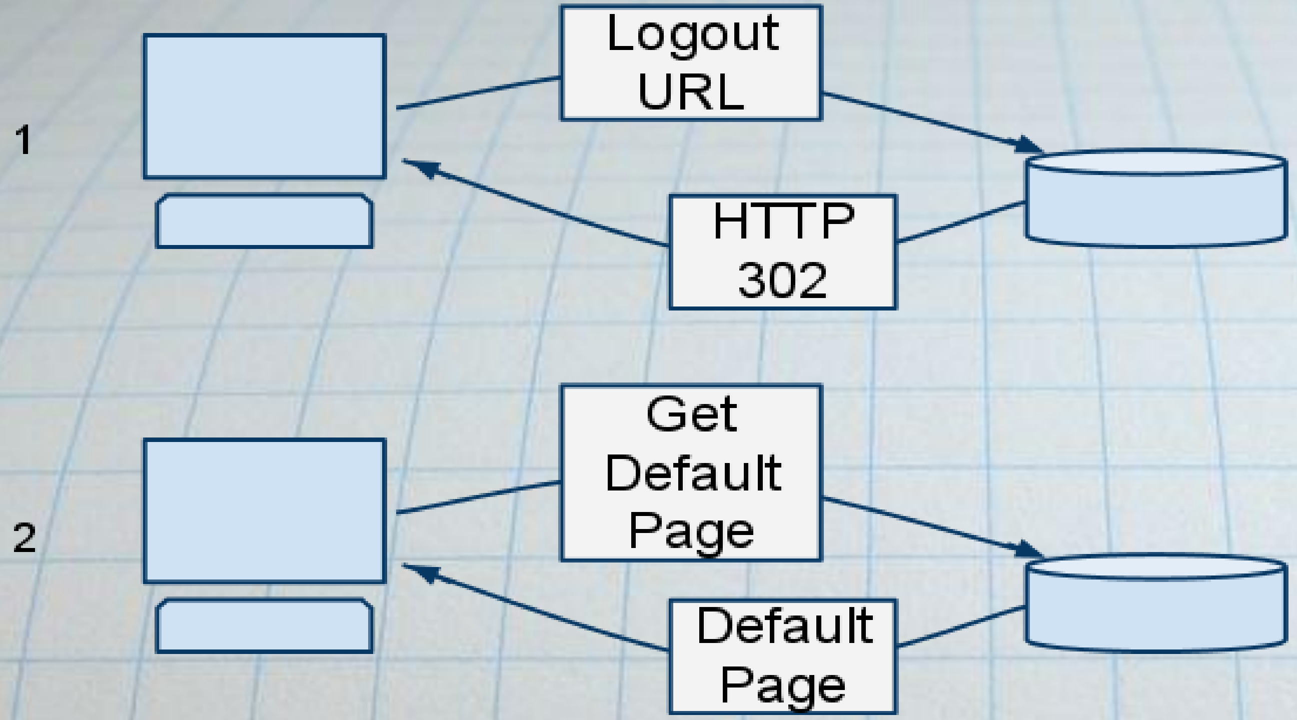
HTTP/1.1 200 OK
Pragma: no-cache
Content-Length: 2150
Keep-Alive: timeout=15, max=100
Content-Type: text/html; charset=iso-8859-1

<html>
... **web page**

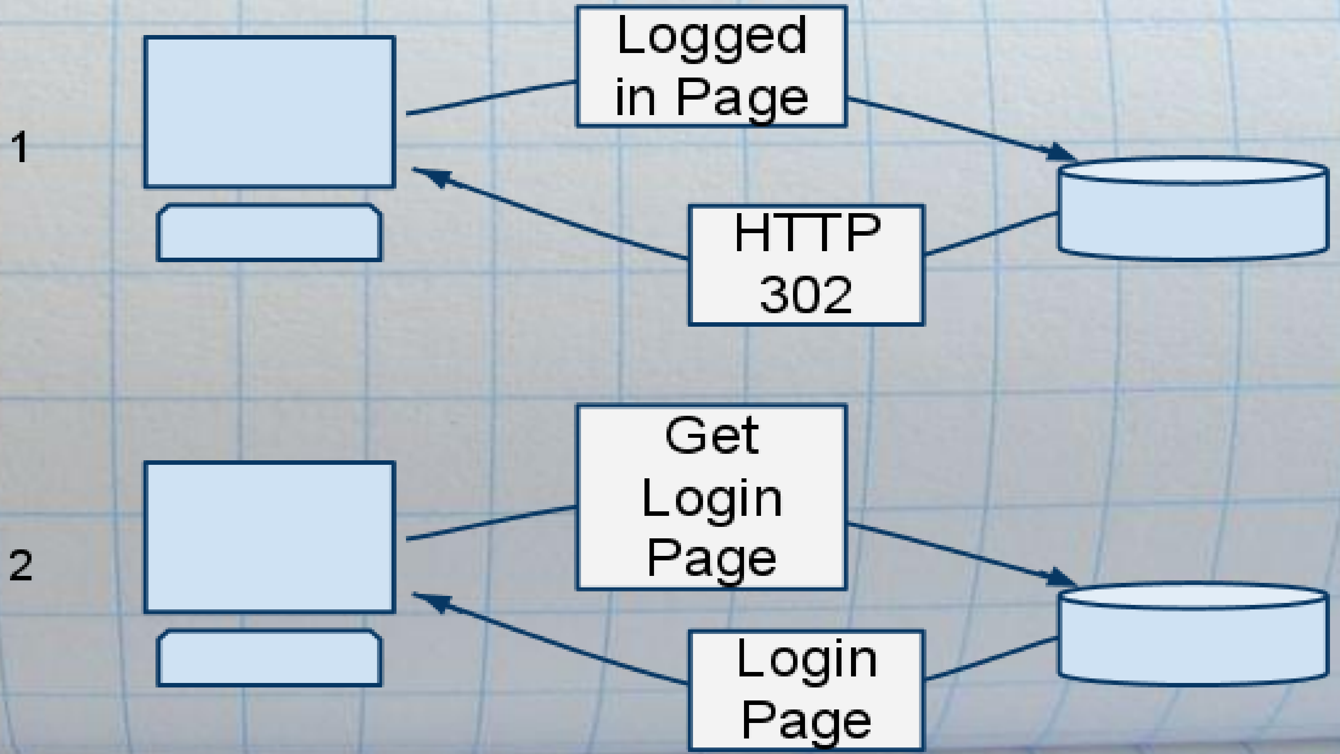
Log Out

- Pre-Login
- Login Page
- Login Redirect
- Logged In
- **Log Out**

- How do users log out:
  - They click on the logout link.
  - Their session expires due to inactivity or absolute timeout.
  - They complete an action.
  - They navigate to a non-logged-in section of the site.
- If the user's session didn't expire, they get a response which contains a Set-Cookie header that expires the logged in cookie and then redirects the user.
- Otherwise they get redirected to the login page.



Clicked on Logout Link



Logged out due to inactivity

Attacks!

The fun stuff.

Attack Goals

- Bypass Login
- Login as another user
- Force logged in users to take actions
- Get logged in users' information
- Affect pre-login actions that affect logged in actions
- Get users to login to a known session or account
- Get valid usernames
- Get valid user passwords
- Get valid user email addresses
- Lockout users

Pre-Login

- **Pre-Login**
- Login Page
- Login Redirect
- Logged In
- Log Out

- SQL Injection - same database
- XSS as a Social Engineering vector
- Carry over attacks:
  - Cookie attacks: XSS, lack of SSL, Header Injection, token prediction
  - Session via URL token (no cookies)
  - CSRF and Clickjacking
- User Enumeration:
  - Password Reset
  - Account Creation
  - Login Form
- Inadequate SSL Coverage
- Combination XSS with CSRF to the logged in section to get sensitive data.

Login Page

- Pre-Login
- **Login Page**
- Login Redirect
- Logged In
- Log Out

- SQL Injection to bypass verification
- XSS as a key logger
- User Enumeration
- Password Bruteforcing
- SQL Injection for password gathering
- Login CSRF
  - Contests
  - Stored data
  - I was framed!
- Inadequate SSL
- Account Lockout

Login Redirect

- Pre-Login
- Login Page
- **Login Redirect**
- Logged In
- Log Out

- Header Injection: Location header
- Session Fixation
- Predictable session token
- Forced redirection
  - Off site (Referer header)
  - CSRF
- Gotta have the SSL
- Javascript or meta tag redirect XSS

Logged In

- Pre-Login
- Login Page
- Login Redirect
- **Logged In**
- Log Out

- XSS framework for full control (BeEF)
- XSS for session token capture
- SQL Injection via CSRF
- CSRF and Clickjacking
- Inadequate SSL coverage
- Authentication bypass
- Disclosure of URL parameters (Referer)
- AJAX hijacking
- Force Logout

Log Out

- Forced redirection
- Header injection: Location
- Session reuse / Inadequate log out
- CSRF logout

- – Pre-Login
- – Login Page
- – Login Redirect
- – Logged In
- – **Log Out**

Conclusions

- Login and Authentication can't be easily segregated from the applications that use it.
- Pre-Login, subdomains, parent domains, and sister domains all can affect the Login and Authentication functionality.
- Pre-Login must either have no session or be under SSL.
- User enumeration protection applies to the Login page as well as Account Creation and Password Reset.
- XSS and SQL Injection are pretty much Game Over.
- Stopping bruteforcing of passwords is difficult, so make the passwords difficult to bruteforce. Password Rules.
- Javascript redirects can lead to DOM based XSS.
- Update the session cookie during the redirection step.
- Use Cryptography for security related tokens.

Conclusions (cont.)

- Watch what goes into the URL. This can get sent off-site in the Referer HTTP header.
- Force users to use cookies. There's no excuse anymore.
- A framework or systematic approach should be taken for Authentication, HTML output, SQL, and CSRF protection.
- AJAX may require CSRF protection for GET requests, too.
- Expiring a session cookie is not a sufficient logout procedure.

Questions?

ben@kedalion-security.com