Web Security:
Session management 2

CS 161: Computer Security

Prof. Raluca Ada Popa

April 13, 2020

Some content adapted from materials by David Wagner or Dan Boneh
Announcements

• Starting recording
• Thanks for feedback
  – slowing down
  – Toby Chen checking chat, specify if question is for professor or for TA
• Project 3 part 1 due Tuesday, April 17 at 11:59pm (extended)
• HMW3b released, due 4/24
• Will release proj 3, part 2, 4/15
• Done grading MT2, need to prepare for regrades
Recall: When browser sends cookie

Browser sends all cookies in URL scope:
• cookie-domain is domain-suffix of URL-domain, and
• cookie-path is prefix of URL-path, and
• [protocol=HTTPS if cookie is “secure”]

Goal: server only sees cookies in its scope
Recall: when browser sends cookie

A cookie with
  domain = example.com, and
  path = /some/path/
will be included on a request to
  http://foo.example.com/some/path/subdirectory/hello.txt
Session management
Sessions

- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long  (Gmail - two weeks)
    or short  (banks)
  - without session mgmt:
    users would have to constantly re-authenticate

- Session management:
  - Authorize user once;
  - All subsequent requests are tied to user for a period
Pre-history: HTTP auth

One username and password for a group of users
HTTP request: GET /index.html
HTTP response contains:
   WWW-Authenticate: Basic realm="Password Required"

Browsers sends hashed password on all subsequent HTTP requests:
   Authorization: Basic ZGFddfizdzsdfgkjhczl1NXRleHQ=
HTTP auth problems

• Hardly used in commercial sites
  – User cannot log out other than by closing browser
    • What if user has multiple accounts?
    • What if multiple users on same computer?
  – Site cannot customize password dialog
  – Confusing dialog to users
  – Easily spoofed
Session token

• A temporary identifier for a user, usually random or cryptographic so that an attacker cannot guess it
• If an attacker gets a session token, it could access the user’s account for the duration of that token
Browser

GET /index.html

Web Site

set anonymous session token

GET /books.html

anonymous session token

POST /do-login

Username & password

elevate to a logged-in session token

POST /purchase

logged-in session token

check credentials

Validate token
Storing session tokens: Lots of options (but none are perfect)

- Browser cookie:
  
  Set-Cookie: SessionToken=fduhye63sfdb

- Embed in all URL links:
  
  https://site.com/checkout?SessionToken=kh7y3b

- In a hidden form field:
  
  `<input type="hidden" name="sessionid" value="kh7y3b">`
Storing session tokens: problems

- Browser cookie:
  - browser sends cookie with every request, even when it should not (CSRF)

- Embed in all URL links:
  - token leaks via HTTP Referer header
  - users might share URLs

- In a hidden form field: short sessions only

Better answer: a combination (1) and (3) above (e.g., browser cookie with CSRF protection using form secret tokens)
Cross Site Request Forgery
HTML Forms

- Allow a user to provide some data which gets sent with an HTTP POST request to a server

```html
<form action="bank.com/action.php">
  First name: <input type="text" name="firstname">
  Last name: <input type="text" name="lastname">
  <input type="submit" value="Submit"></form>
```

When filling in Alice and Smith, and clicking submit, the browser issues

HTTP POST request

```
bank.com/action.php?firstname=Alice&lastname=Smith
```

As always, the browser attaches relevant cookies
Consider the cookie stores the session token

- Server assigns a random session token to each user after they logged in, places it in the cookie
- The server keeps a table of [username -> session token], so when it sees the session token it knows which user
- When the user logs out, the server clears the session token
Session using cookies

**Browser**

```
POST/login.cgi
```

**Set-cookie: session token**

**Server**

```
GET/POST...
Cookie: session token
```

**response**
CSRF Attack Basic Picture

1. Establish session
2. Visit server
3. Receive malicious page
4. Send forged request (w/ cookie)

What can go bad? URL contains transaction action
Cross Site Request Forgery (CSRF)

– User logs in to bank.com
  • Session cookie remains in browser state

– User visits malicious site containing:
  
  <form name=F action=http://bank.com/BillPay.php>
    <input name=recipient value=badguy> …
    <script> document.F.submit(); </script>

– Browser sends user auth cookie with request
  • Transaction will be fulfilled

• **Problem:**
  – cookie auth is insufficient when side effects occur
Form post with cookie

www.attacker.com → Victim Browser → www.bank.com

GET /blog HTTP/1.1
Form post with cookie

www.attacker.com  →  GET /blog HTTP/1.1  →  www.bank.com

form action=https://www.bank.com/transfer
method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>

POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E

HTTP/1.1 200 OK
Transfer complete!

User credentials
IS THE PACE SLOW ENOUGH?
An attacker could

• add videos to a user’s "Favorites,"
• add himself to a user’s "Friend" or "Family" list,
• send arbitrary messages on the user’s behalf,
• flagged videos as inappropriate,
• automatically shared a video with a user’s contacts, subscribed a user to a "channel" (a set of videos published by one person or group), and
• added videos to a user’s "QuickList" (a list of videos a user intends to watch at a later point).
Popular websites fall victim to CSRF exploits
Defenses

ideas?
CSRF Defenses

• CSRF token

• Referer Validation

• Others (e.g., custom HTTP Header) we won’t go into
1. goodsite.com server wants to protect itself from CSRF attacks, so it includes a secret token into the webpage (e.g., in forms as a hidden field)

2. Requests to goodsite.com include the secret

3. goodsite.com server checks that the token embedded in the webpage is the expected one; reject request if not

Can the token be?

- 123456
- Dateofbirth

CSRF token must be hard to guess by the attacker
How the token is used

- The server stores state that binds the user's CSRF token to the user's session id
- Embeds CSRF token in every form
- On every request the server validates that the supplied CSRF token is associated with the user's session id
- Disadvantage is that the server needs to maintain a large state table to validate the tokens.