Secure Programming

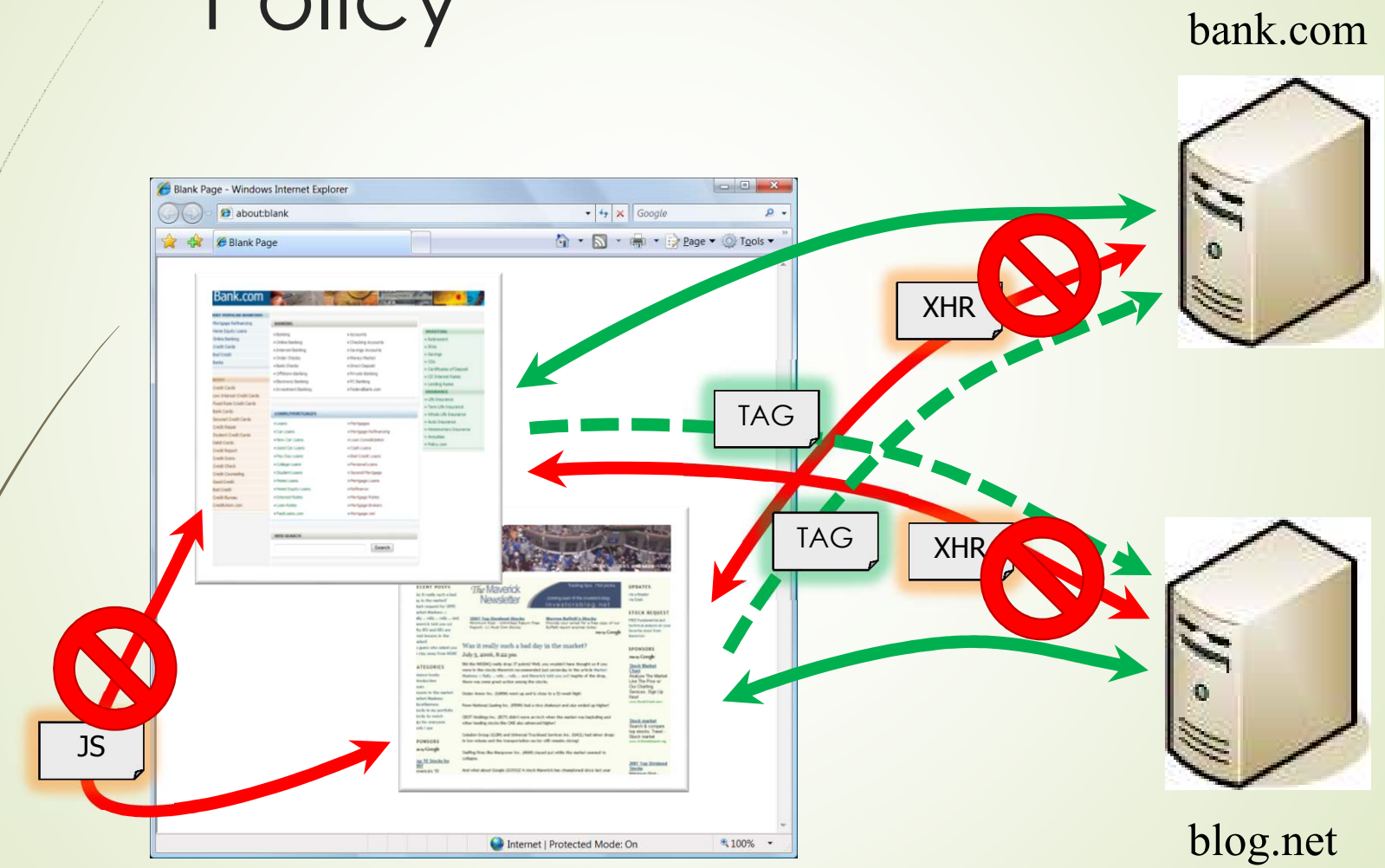
Cross-Site Request Forgery(CSRF) Vulnerabities

Ahmet Burak Can

Agenda

- Web Application Authentication
- CSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion

The Browser "Same Origin" Policy



Explicit Authentication

- The authentication credentials are communicated by the web application
  - URL rewriting: Session token is included in every URL
  - Form based session tokens

Immune against CSRF

(actually only almost immune)

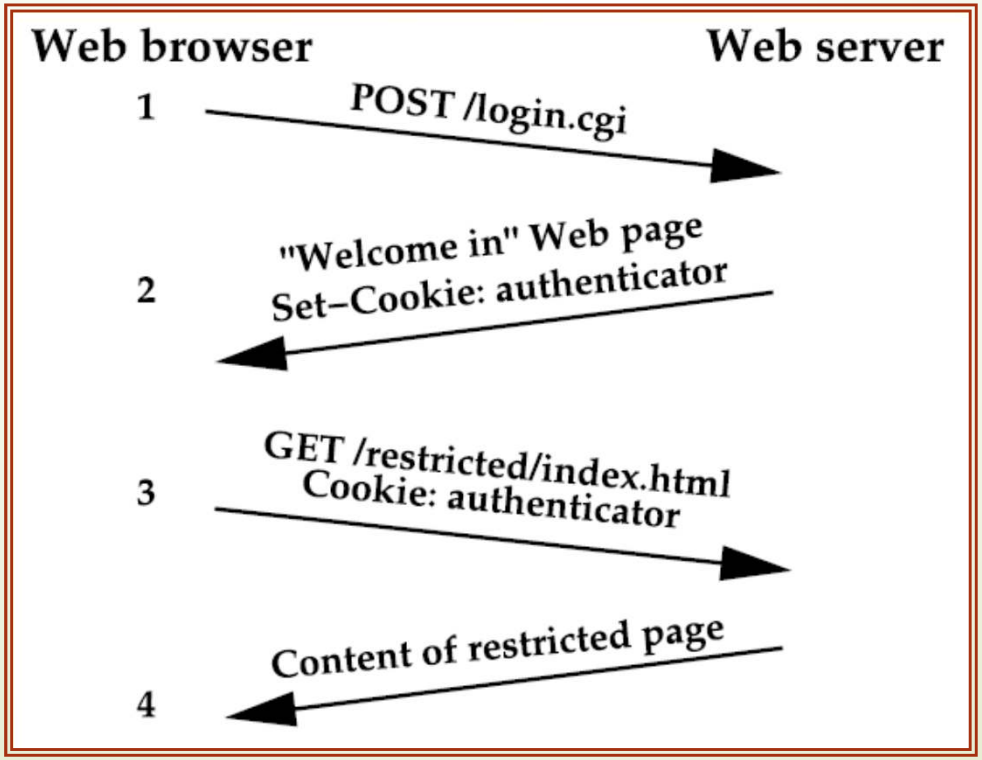
Implicit Authentication

- Automatically executed by the browser
  - Cookies
  - http authentication (Basic, Digest, NTLM)
  - IP based schemes
  - Client side SSL

Potentially vulnerable to CSRF

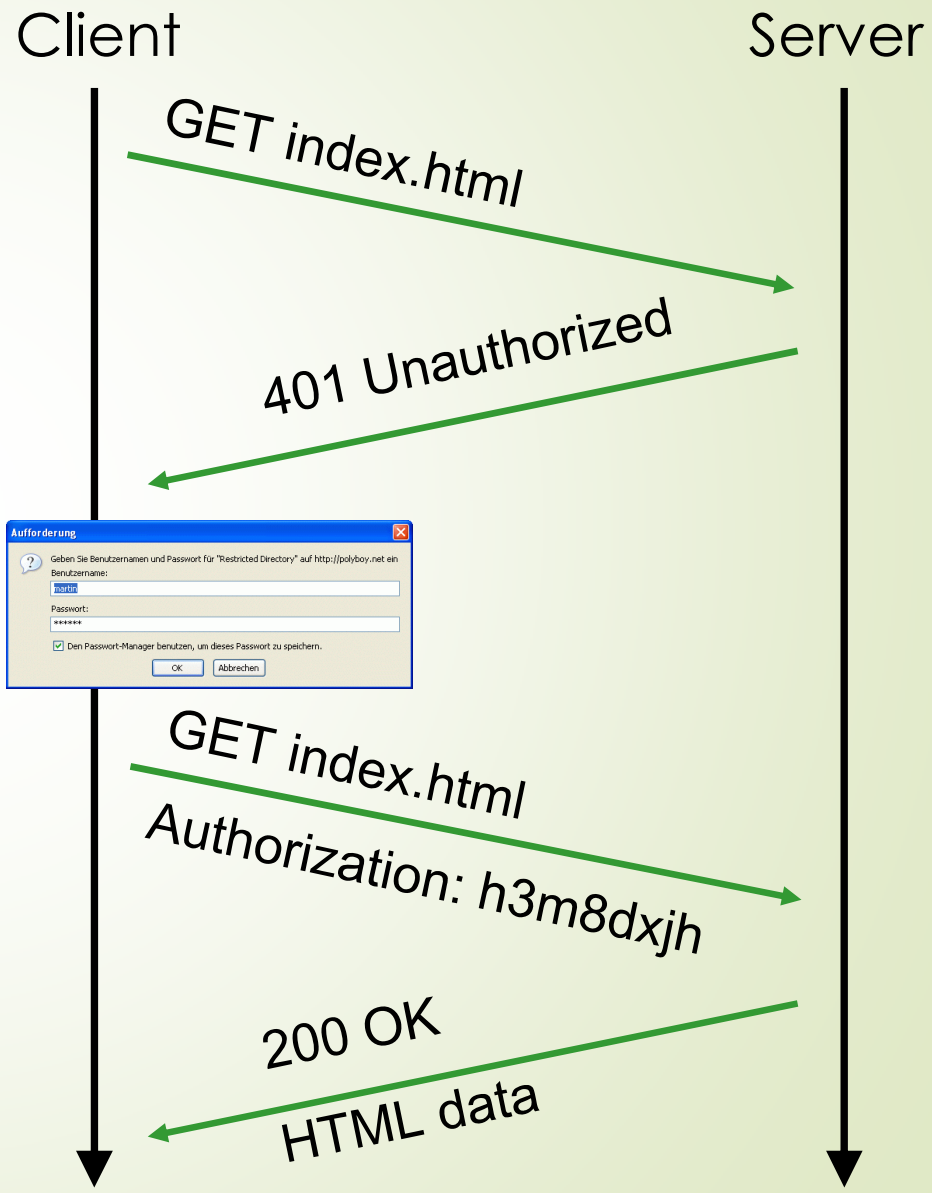
Session management with cookies

- After the authentication form the server sets a cookie at the client's browser
- As long as this cookie is valid, the client's requests are treated as authorized

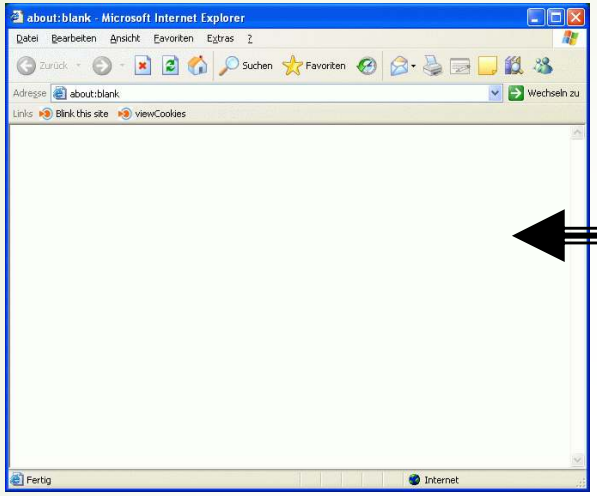


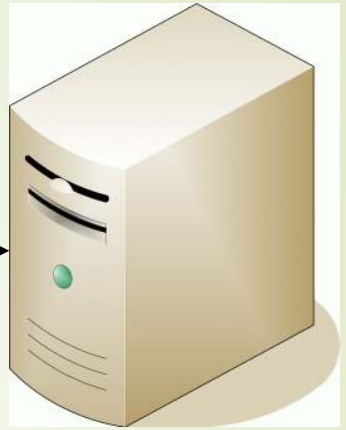
HTTP authentication (Basic, Digest)

- The client requests a restricted resource
- The server answers with a “401 Unauthorized” response
- This causes the client’s browser to demand the credentials
- The client resends the request
- The user’s credentials are included via the “Authorization” header
- Every further request to that authentication realm contains the credentials automatically



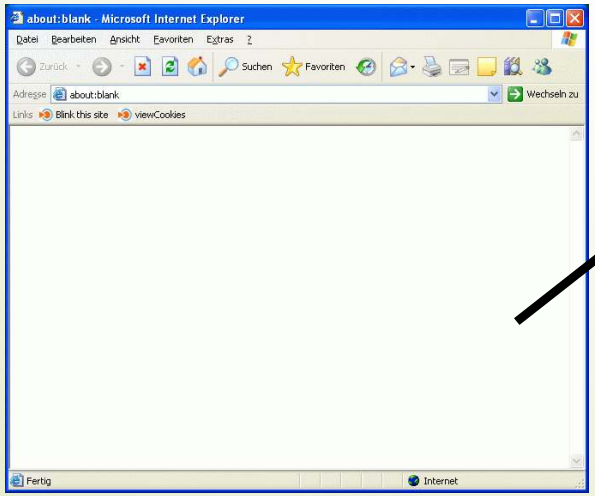
IP based authentication





Intranet webserver

Firewall



Client side SSL authentication

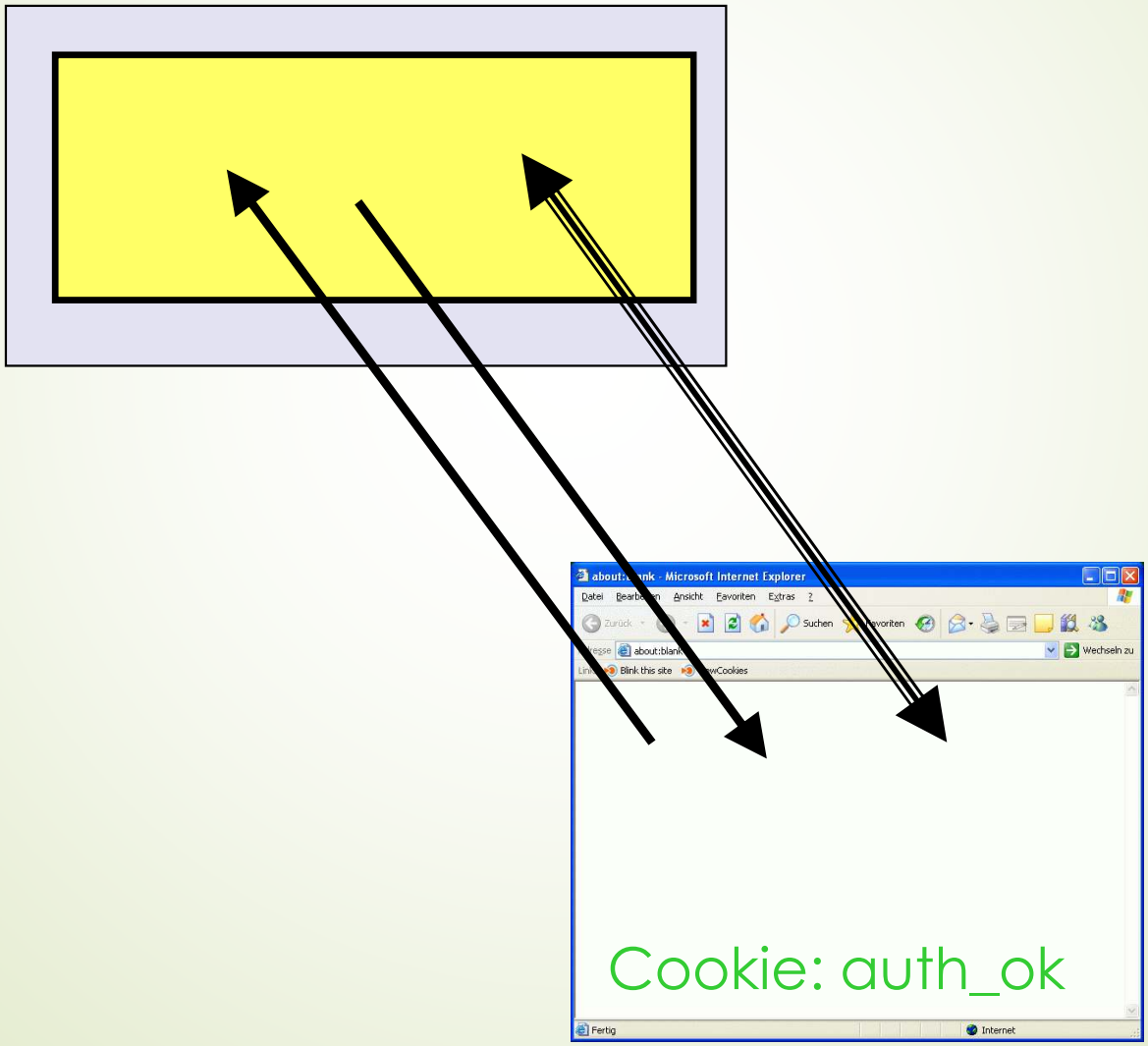
- The client web browser possesses a X.509 certificate that was signed by an authority that is trusted by the web application
- Initial authentication:
  - The client has to prove his identity
  - For this reason, the web server demands a valid signature from the client
  - → "SSL handshake"
  - Depending on the browser, the user may or may not confirm the initial handshake entering a password (only once)
- If the handshake was successful, a SSL session is established between the cli browser and the web server
- As long as the SSL session is valid, all request to the web server are transmitte using the negotiated credentials

CSRF / Session Riding

- Exploits implicit authentication mechanisms
  - Known since 2001
  - CSRF a.k.a. CSRF a.k.a. “Session Riding” (a.k.a. “Sea Surf”)
  - Unknown/underestimated attack vector (compared to XSS or SQL injection)
- The Attack:
  - The attacker creates a hidden http request inside the victim’s web browser
  - This request is executed in the victim’s authentication context

CSRF / Session Riding (II)

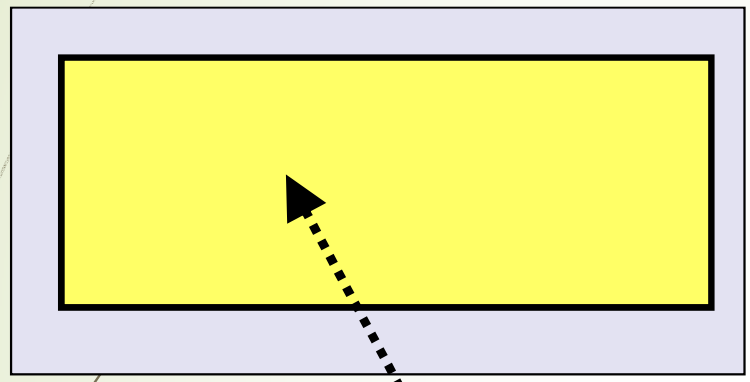
www.bank.com

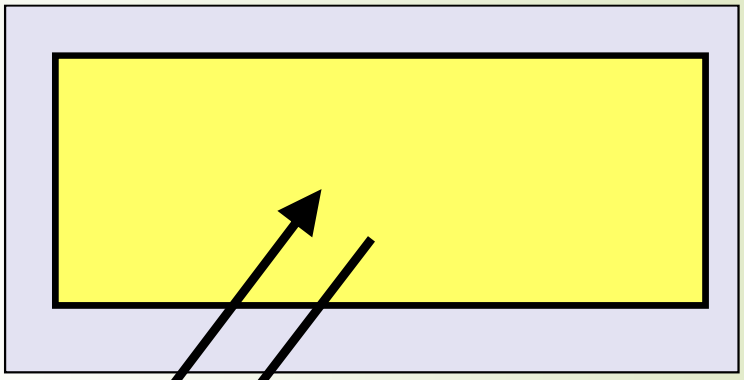


CSRF / Session Riding (II)

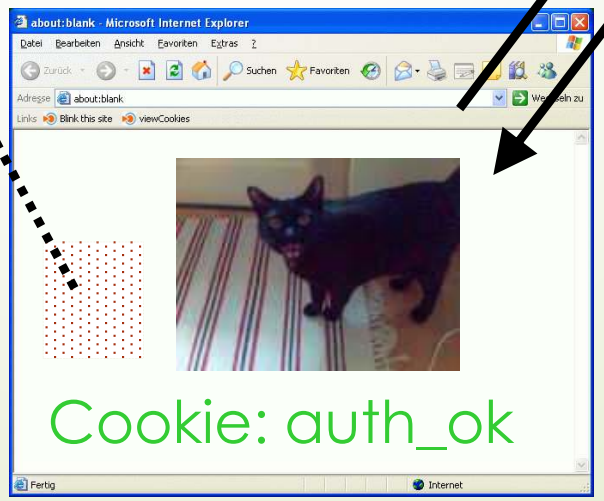
www.bank.com

www.attacker.org

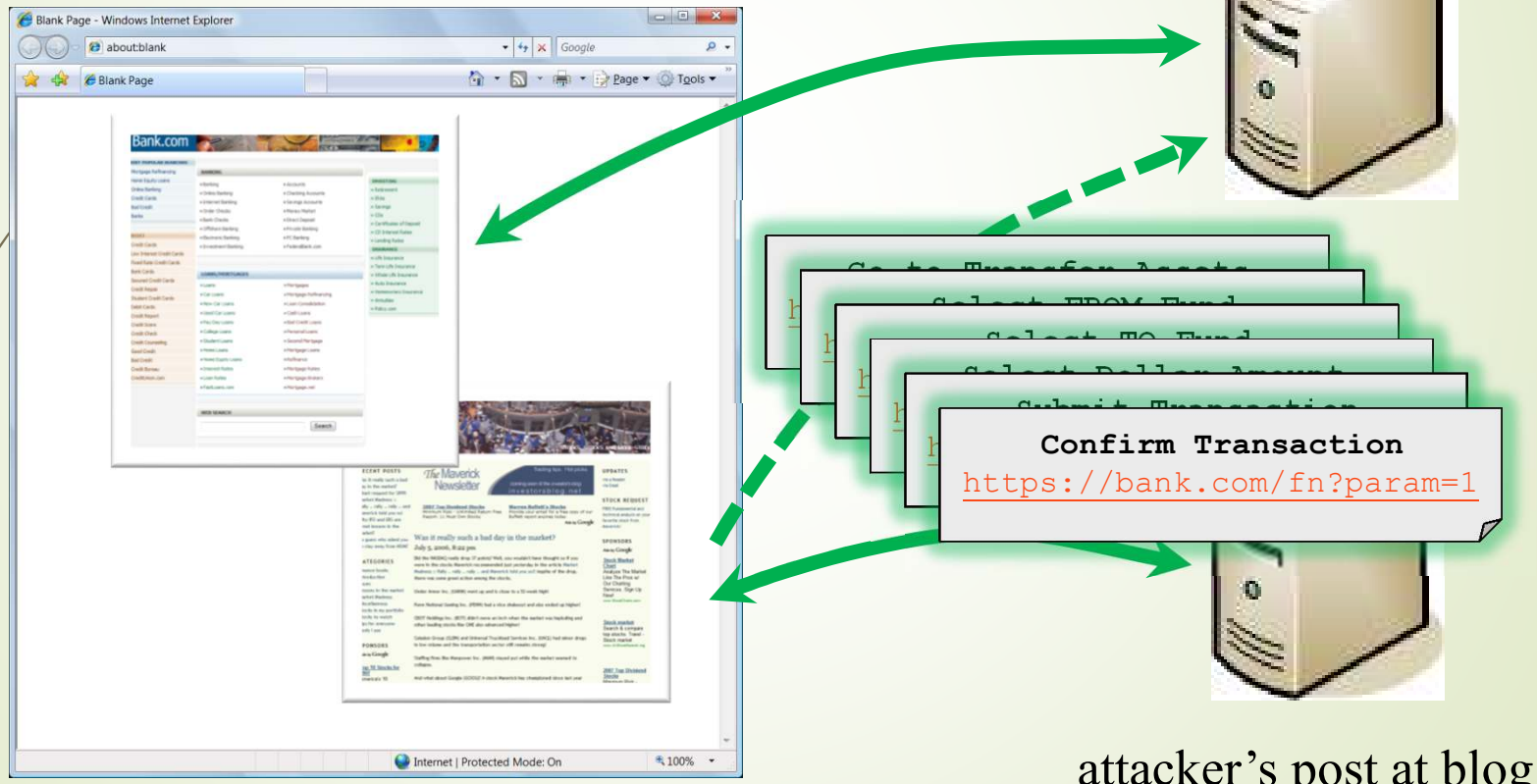




GET transfer.cgi?am=10000&an=3422421



Cross-Site Request Forgery



How Does CSRF Work?

- Tags

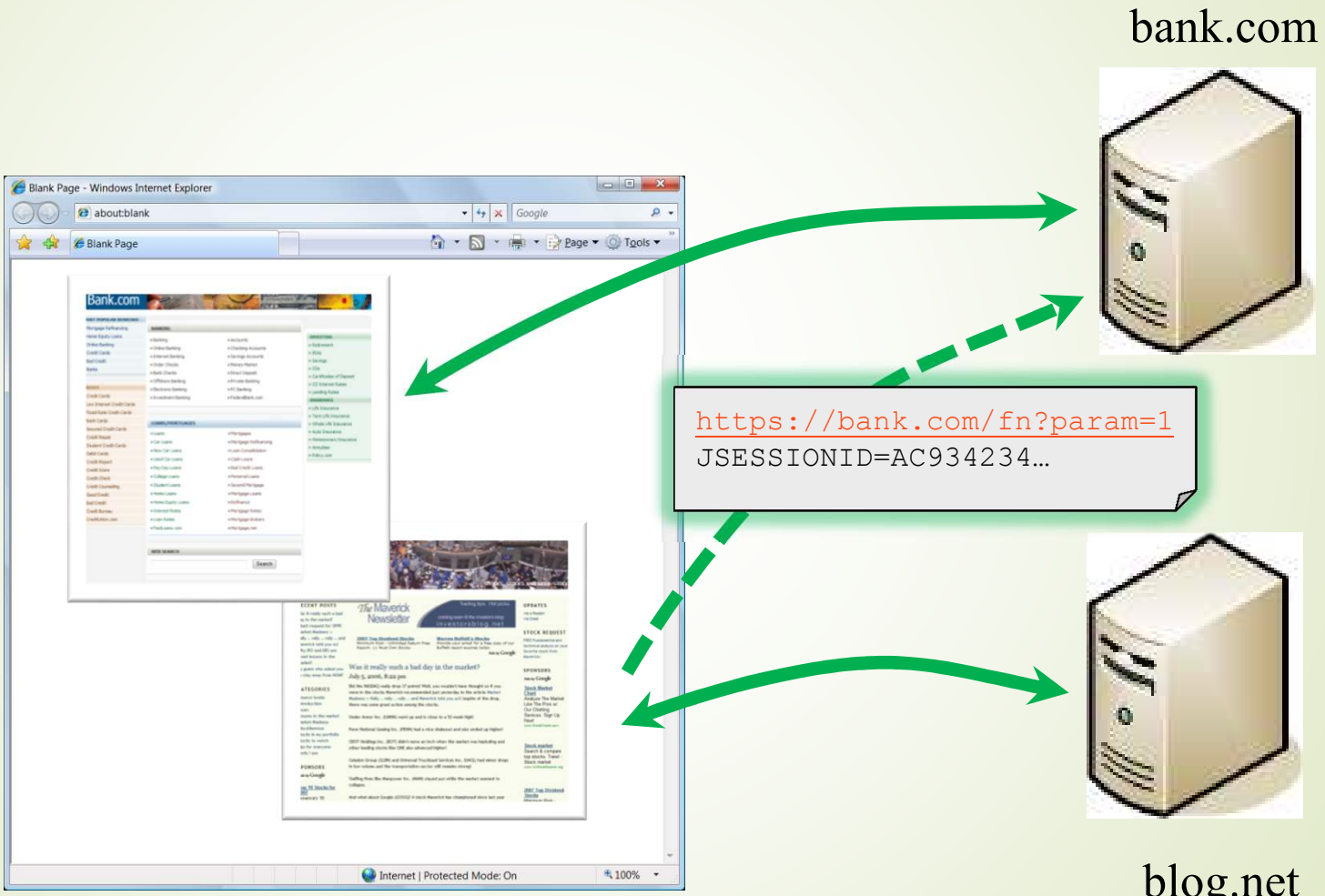
```
<img src="https://bank.com/fn?param=1">
<iframe src="https://bank.com/fn?param=1">
<script src="https://bank.com/fn?param=1">
```

- Autoposting Forms

```
<body onload="document.forms[0].submit()">
<form method="POST" action="https://bank.com/fn">
   <input type="hidden" name="sp" value="8109"/>
</form>
```

- XmlHttpRequest
  - Subject to same origin policy

Credentials Included



CSRF / Session Riding (III)

- Cause: The web application does not verify that state changing request were created “within” the web application
- Attack methods:
  - Forging GET requests:
    - Image tag with SRC attribute that points to a state changing URL
    - The URL might be obfuscated by a http redirect
  - Forging POST request:
    - Attacker creates an IFRAME (or a pop-up window)
    - The frame is filled with a HTML form
    - This form is submitted via JavaScript

Cross-domain interactions

- Recall…
  - <script src=http://good.com/foo></script> in bad page would cause legitimate script to run in context of bad page!
  - Instead, malicious page can initiate a POST request to legitimate page, with arbitrary parameters
  - Due to the way web authentication is handled (i.e., using a *cached* credential), http requests will look as if they come from the legitimate user if they are logged in when they view the malicious page

CSRF Example

1. Alice's browser loads page from **bad.com**

2. Script runs causing **evilform** to be submitted with a password-change request by loading **www.good.com/update_pwd** with attacker-specified field

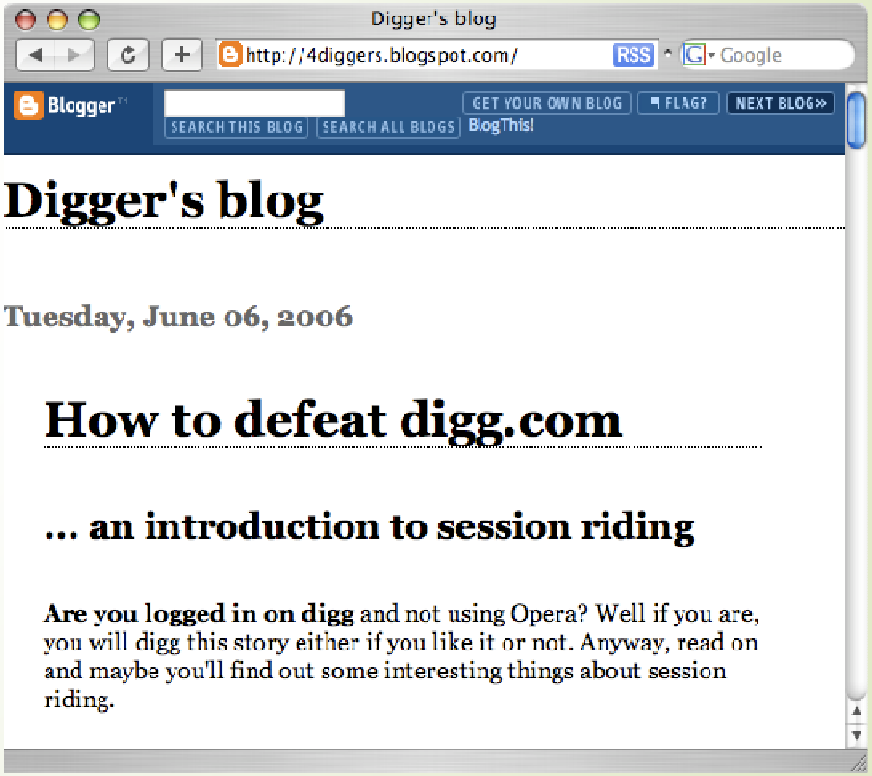
evilform

```
<form method="POST" name="evilform" target="hiddenframe"
  action="https://www.good.com/update_pwd">
  <input type="hidden" id="password" value="badpwd">
</form>
<iframe name="hiddenframe" style="display: none">
</iframe> <script>document.evilform.submit();</script>
```

3. Browser sends authentication cookies to good server. Honest user's password is changed to **badpwd**!

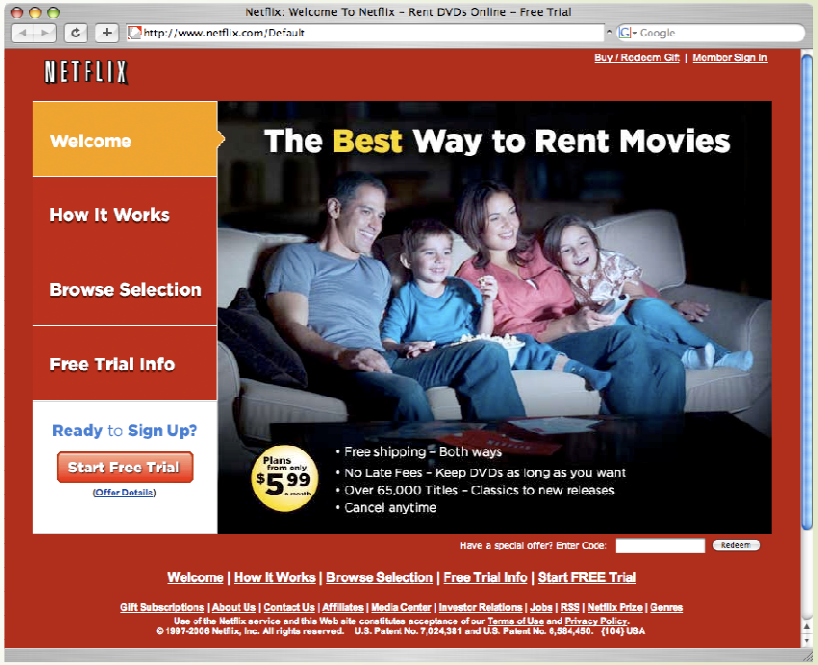
Example 1: Breaking Applications

- Vulnerable: digg.com
  - digg.com's frontpage is determined by the number of "diggs" a certain story gets
  - Using CSRF a webpage was able to cause the victim's browser to "digg" an arbitrary URL
  - The demo page "digged" itself



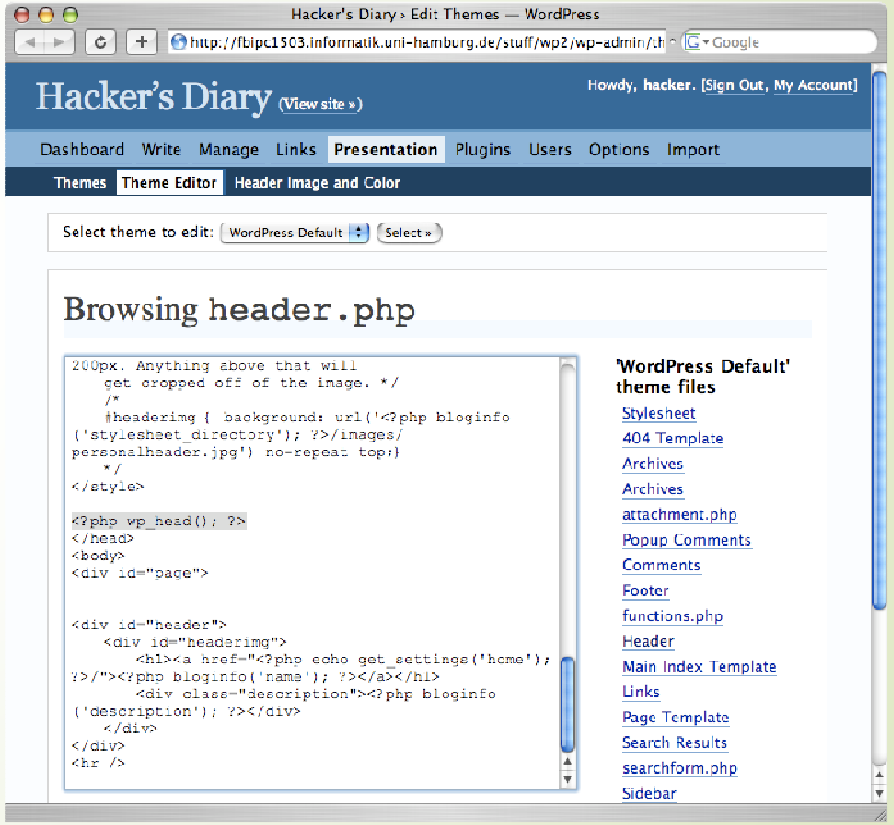
Example 2: Causing Financial Loss

- Vulnerable: Netflix.com
  - Add movies to your rental queue
  - Add a movie to the top of your rental queue
  - Change the name and address on your account
  - Change the email address and password on your account (i.e., takeover your account)
  - Cancel your account (Unconfirmed/Conjectured)



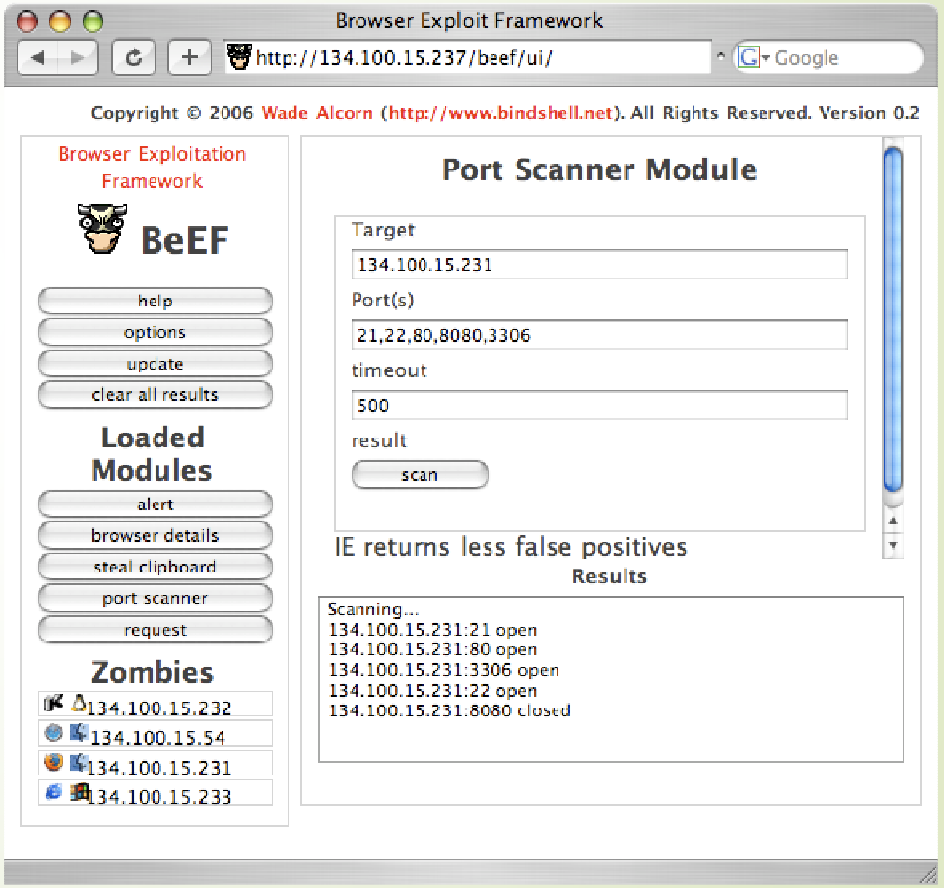
Example 3: Owning the Server

- Vulnerable: Wordpress 2.02
  - Wordpress' theme editor was susceptible to CSRF
  - Wordpress theme-files can be php-files
  - Via CSRF an attacker could modify those files to contain arbitrary php-code



Example 4: Exploring the Intranet

- Vulnerable: (most) intranet webservers
  - By dynamically including external images and using timed JavaScript events, a malicious website can, e.g.:
    - Portscan the intranet
    - Fingerprint existing web servers and installed applications
- → "JavaScript Malware"



CSRF / Session Riding (IV)

- General problem:
  - Session Riding vulnerabilities are NOT caused by programming mistakes
  - Completely correct code can be vulnerable
  - The reason for Session Riding lies within http:
    - No dedicated authentication credential
    - State-changing GET requests
    - JavaScript

"Preventing Session Riding"
is actually
"fixing the protocol"

Preventing CSRF attacks

- Inspect referrer headers
  - HTTP protocol specifies a header indicating the URL of the document from which current request originated
- So **good.com** can try to prevent CSRF attacks by ignoring POST requests if the referrer is not **good.com**
- However…
  - Referrer fields can be absent for legitimate reasons (e.g., new window; stripped by proxies)

Misconceptions

- Referrer checking
  - Some users prohibit referrers
    →referrerless requests have to be accepted
  - Techniques to selectively create http request without referrers exist:

Method/Browser	IE 5	IE 6*	IE 7**	FF 1.07	FF 1.5	O 8	S 1.2
META Refresh				X	X		
Dynamic filled frame	X	X	X	X	X		X
Pop up window (regular)	X	X	X				
Pop up window (dynamically filled)				X	X		

IE: Internet Explorer; FF: Firefox; S: Safari; O: Opera; *: IE 6 XPSP 2; **: IE 7 (Beta 2)

Table 1. Generating referrerless requests ("X" denotes a working method)

  - Furthermore, referrers can be spoofed with Flash

Complete mediation

- Prevent CSRF attacks by requiring user re-authentication
- Not practical to do this all the time
  - User will be come frustrated!
- Can require for ‘high-value’ transactions

Client-side protection

- (Assumes servers do not use GET requests for modifying data)
- Browser plug-in that filters out POST requests unless requesting site and target site satisfy same-origin policy
  - Might still filter out some legitimate requests

Server-side protection

- Prevent CSRF attacks by allowing the legitimate server to distinguish links in 'fresh' pages it serves, from links embedded in attacker pages
- Add authenticated "action token" as hidden field in pages served; check token upon POST request
  - Same-origin policy prevents 3rd parties from reading the token

Action tokens

- Need a way to *bind* token to session
  - At beginning of session, send cookie with random session-id to user
  - Compute MAC over the URL and the cookie (note that cookie will be sent in any subsequent requests)
- This is potentially vulnerable to XSS attacks
  - Attacker injects script that steals user's cookie and token