Network and Web Security

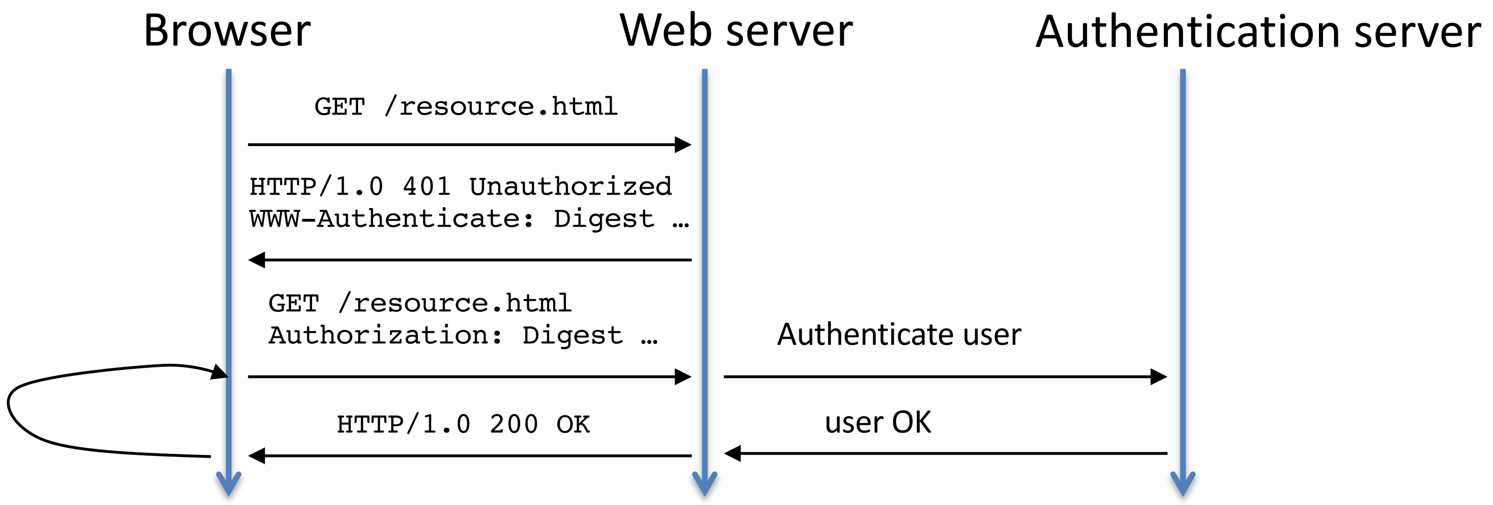
Secure sessions

Dr Sergio Maffeis
Department of Computing
Course web page: https://331.websec.fun

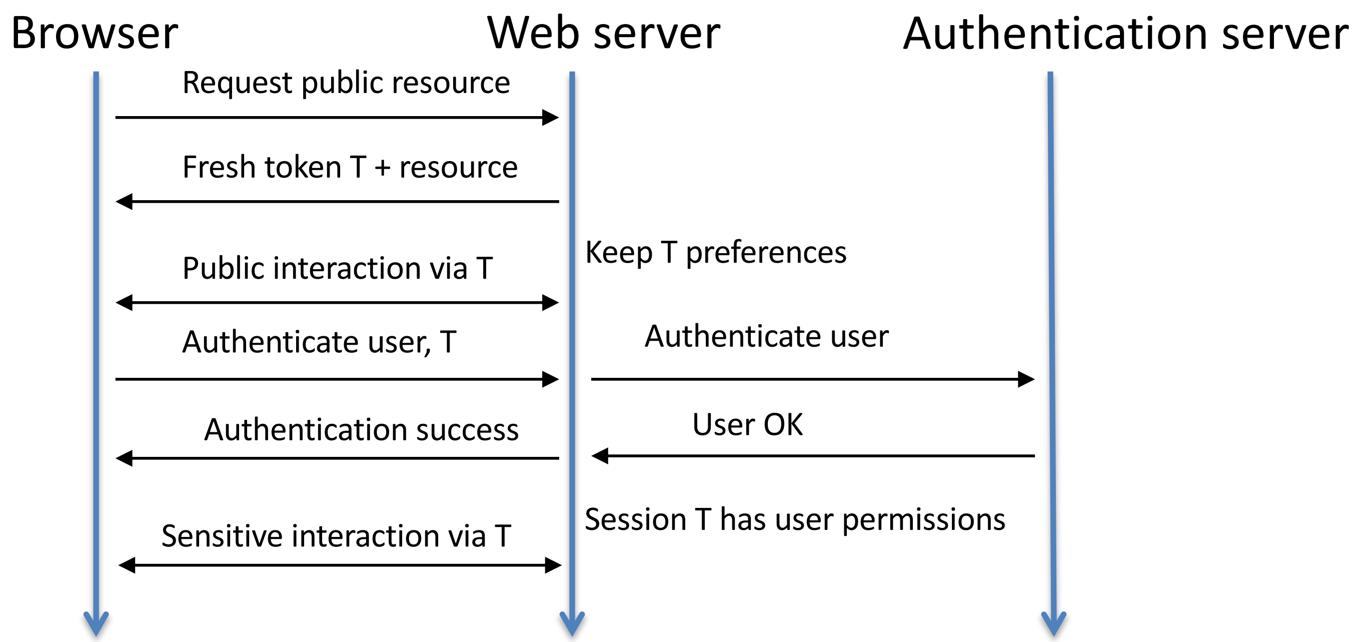
HTTP Authentication

Imperial College London

- HTTP is a stateless protocol
  - Each time a user needs to do an action requiring authorization, its identity needs to be established anew
- HTTP Basic Authentication
  - Send username and password in clear text
  - Wise to use at least HTTPS
  - Essentially deprecated
- HTTP Digest Authentication
  - Send hash of password and server-generated nonce that may restrict validity
    - Time stamp, client IP, etc.
  - Does not protect other fields or headers
- Limitations of HTTP authentication
  - Inefficient: contact the authentication server at every request
  - Cumbersome: user needs to close browser to sign out
  - Annoying: user needs to re-authenticate for each different web asset
- Security issues
  - Credentials sent on the wire with every request
  - Password dialogue easy to spoof and confusing for user
  - MITM can tamper with Digest nonce and launch offline dictionary attack



Sessions

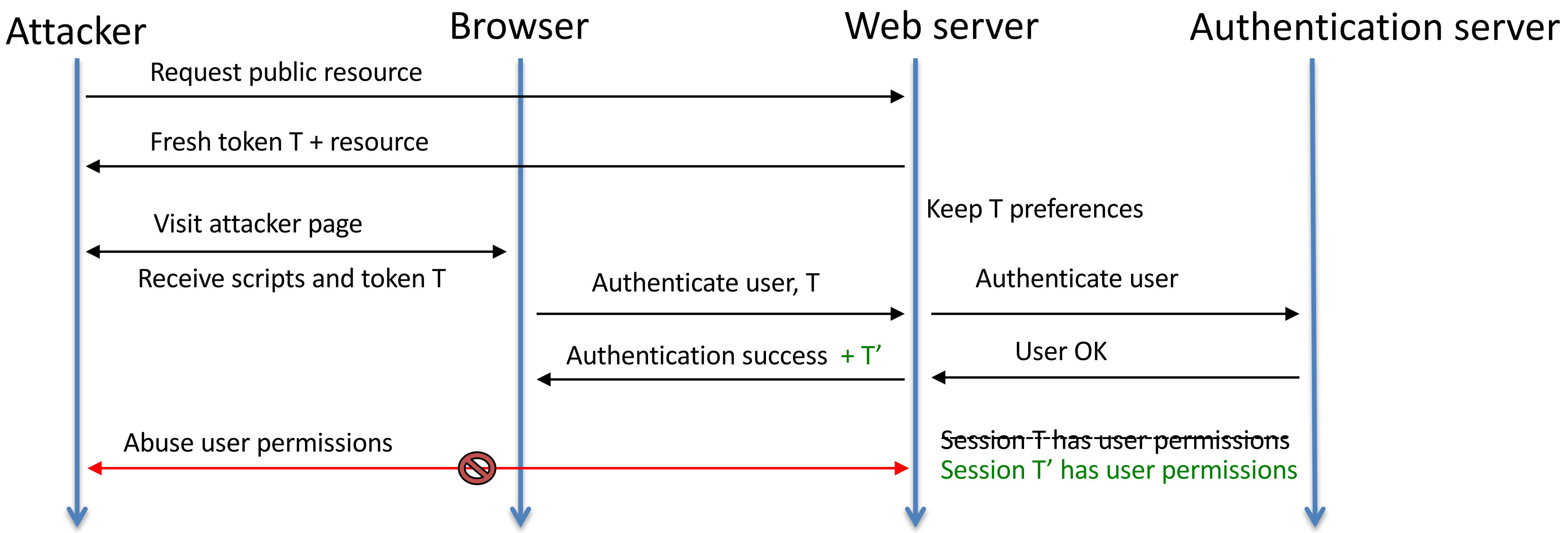


- Unauthenticated sessions
  - The server issues a short-lived token to the client
  - The client presents the token with request that affect client state on the server
  - Useful to keep track of web app state on behalf of anonymous user
- Authenticated sessions
  - The client authenticates once
  - The client presents the token when authorization is needed
  - More efficient, flexible, and complicated than HTTP Authentication
- Session token
  - Also know as session id, `SID`, `SSID`, `PHPSESSIONID`, …
  - Typically implemented as **cookies**
- Most servers provide modules to support sessions and handle session tokens

Attacks: session fixation

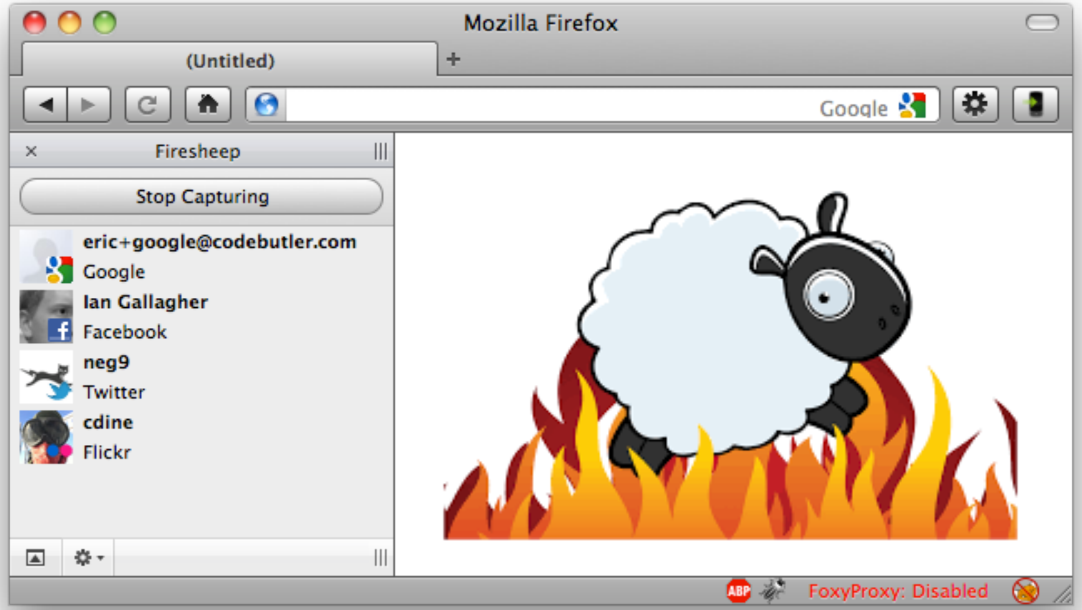
Imperial College London

- Naïve session implementations are subject to session fixation attacks
- Attacker obtains unauthenticated session token by connecting to web server
- Tricks user to log in using attacker's token
  - For example using XSS or MITM
- After login, the token is associated to a valid session
  - Elevation of privilege: attacker can use token to perform authorized actions on behalf of the user
- Countermeasure: after login issue a new token



Attacks: session hijacking

- Attacker obtains a valid token and performs sensitive actions on behalf of user
  - Guessing attack
  - MITM: steal over HTTP connection and WiFi
    - Possible also when HTTP is used only **after** logging in over HTTPS
  - XSS attack
- Mitigations
  - Send session tokens only over HTTPS
  - Invalidate session on server after logout
    - Restricts window-of-opportunity for attacker that has stolen a token
  - Use secure tokens
- Firesheep extension for Firefox: PoC session hijacking on Facebook, Twitter, etc.



Secure tokens

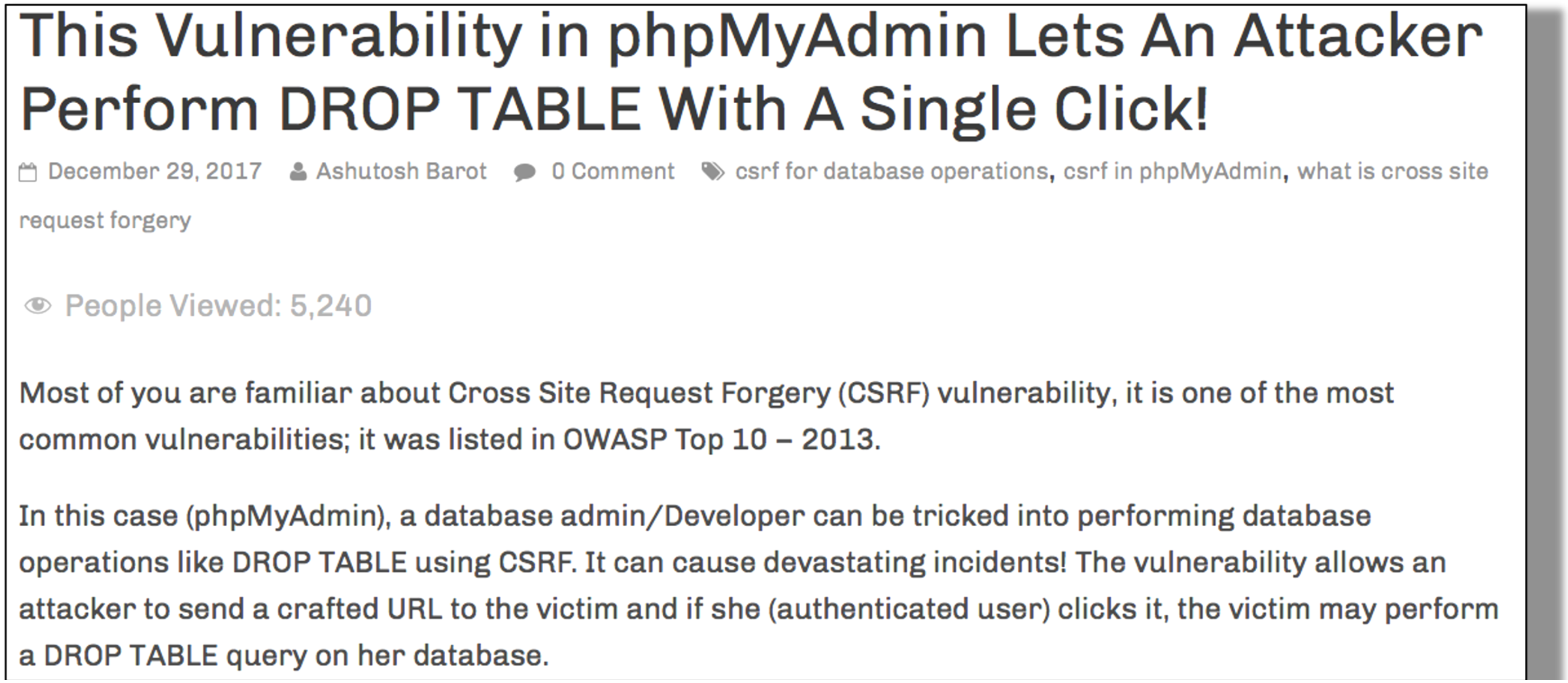
Imperial College London

- Tokens can be spoofed
  - Make tokens unpredictable using randomness
- Tokens can be stolen
  - Restrict where attacker can use them
    - Bind session to client-context such as IP address, SSL session Id, browser fingerprint
  - But…
    - User may get logged out unexpectedly
      - IP changes when switching from WiFi to Ethernet
      - SSL session Id changes when user re-open website with existing session
    - Website attacker can often use victim browser
- Secure token example
  - Session *data* = (timestamp, random value, user id, login status, client-context)
  - Option 1: server keeps data
    - Small token: MD5(*data*)
    - Overhead of database lookup for each request
  - Option 2: server sends data to client
    - Larger token: Encrypt-then-MAC(*data*)
    - Server must still keep track of login status

Attacks: CSRF

Imperial College London

- Cross-Site Request Forgery (CSRF) exploits trust between browser and target
  - User is in position to issue requests that cause side-effects
    - User logged-in to a web application
    - IP-based access control in a LAN
- Easy to deploy
  - Attacker tricks user into issuing request that causes side-effects unintended by the user
  - Enough for user to visit malicious web page or click on link crafted by the attacker
- Widespread: last 3 months: 139 new CSRF-related CVEs



This Vulnerability in phpMyAdmin Lets An Attacker Perform DROP TABLE With A Single Click!

December 29, 2017 Ashutosh Barot 0 Comment csrf for database operations, csrf in phpMyAdmin, what is cross site request forgery

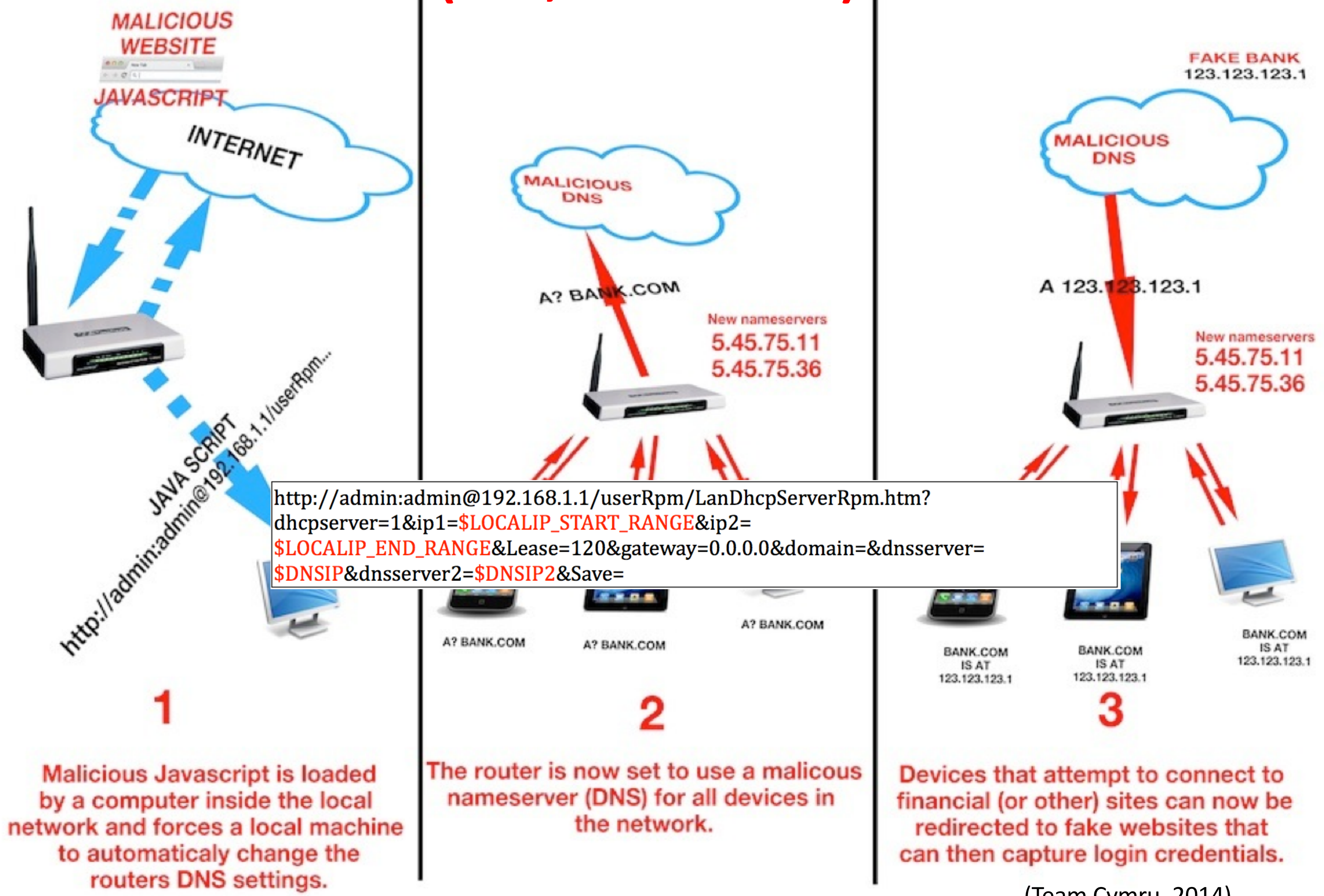
People Viewed: 5,240

Most of you are familiar about Cross Site Request Forgery (CSRF) vulnerability, it is one of the most common vulnerabilities; it was listed in OWASP Top 10 – 2013.

In this case (phpMyAdmin), a database admin/Developer can be tricked into performing database operations like DROP TABLE using CSRF. It can cause devastating incidents! The vulnerability allows an attacker to send a crafted URL to the victim and if she (authenticated user) clicks it, the victim may perform a DROP TABLE query on her database.

CSRF SOHO ROUTER ATTACK

(300,000 victims)



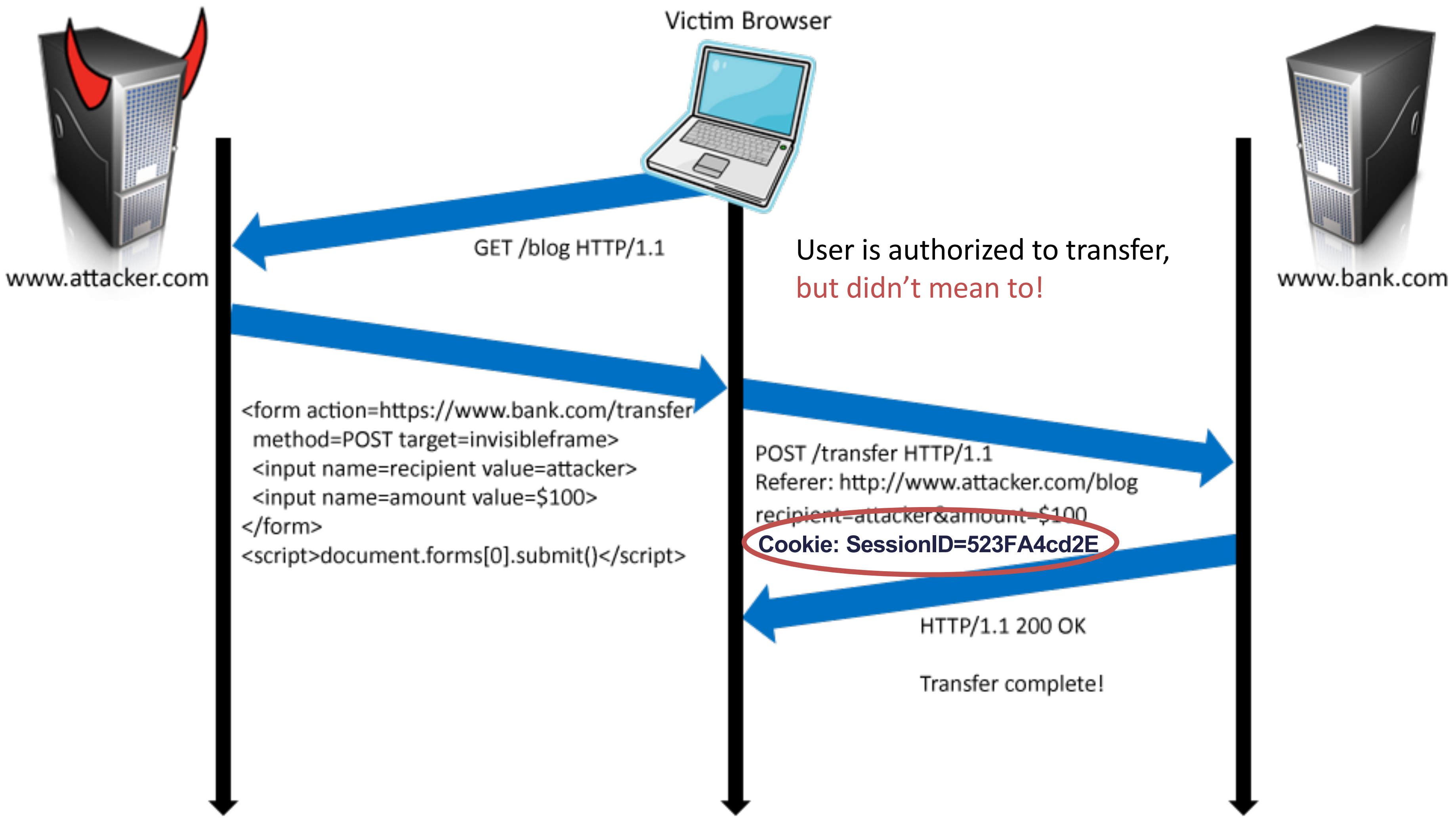
```
http://admin:admin@192.168.1.1/userRpm/LanDhcpServerRpm.htm?
dhcpserver=1&ip1=$LOCALIP_START_RANGE&ip2=
$LOCALIP_END_RANGE&Lease=120&gateway=0.0.0.0&domain=&dnsserver=
$DNSIP&dnsserver2=$DNSIP2&Save=
```

1. Malicious Javascript is loaded by a computer inside the local network and forces a local machine to automaticaly change the routers DNS settings.
2. The router is now set to use a malicous nameserver (DNS) for all devices in the network.
3. Devices that attempt to connect to financial (or other) sites can now be redirected to fake websites that can then capture login credentials.

(Team Cymru, 2014)

Attacks: session CSRF

Imperial College London



User is authorized to transfer,
but didn't mean to!

```
<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>
```

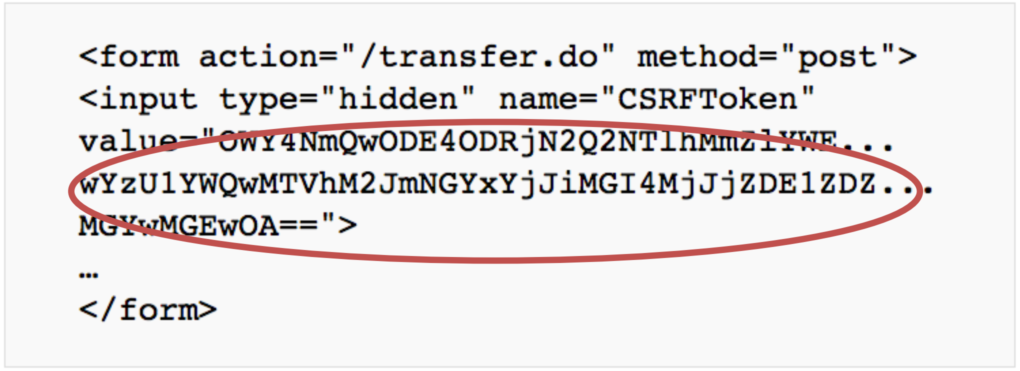
```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

(Mitchell, 2008)

CSRF mitigations

Imperial College London

- Use POST and not GET for sensitive, state changing actions
  - POST body does not leak via `referer` header
  - POST body is not sent in redirections
- Embed a second token as a hidden field of each form presented on authenticated pages
  - The request from the attacker will have the cookie, but not this second token
- Option 1: double cookie
  - Use the same token in form and in cookie, server checks if they are the same
- Option 2: use different tokens in form and in cookie
  - Server knows which 2 should correspond
  - More secure and flexible: form token can be different for each form
  - Could be hash of session ID and intended action to save space on server
- Use SameSite attribute for session cookie
  - Restricts functionality: cannot access existing session via external link
    - Example: page from https://a.com with link to https://github.com/331/privateProject
  - Still not widely adopted, stress-tested
- Many frameworks offer built-in CSRF protections



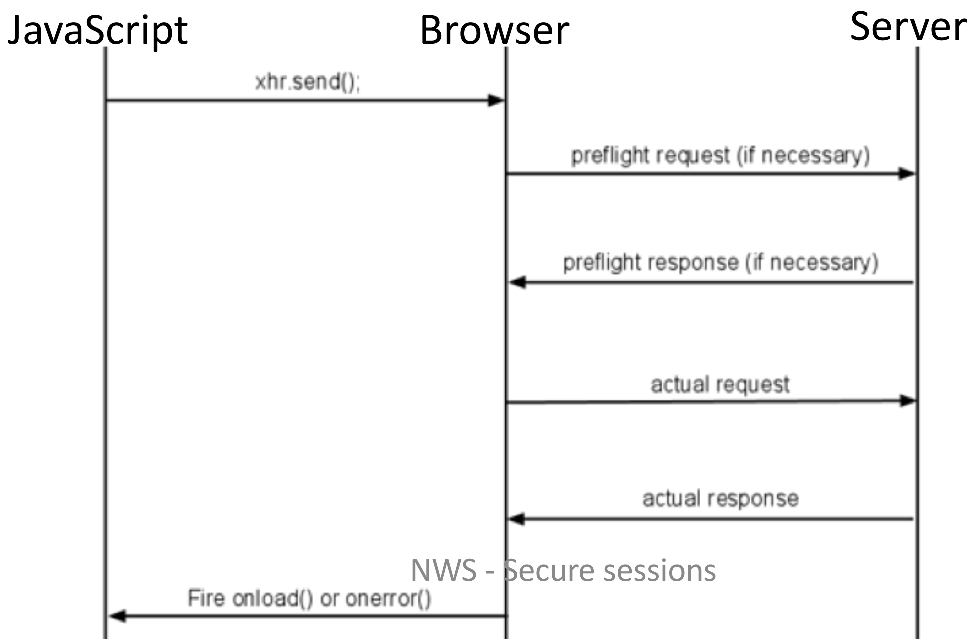
```
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWE...
wYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZ...
MGYwMGEwOA==">
…
</form>
```

Attacker does not know what value to use in the spoofed form!

CORS

Imperial College London

- SOP allows cross-origin communication *when both parties are willing to engage*
  - Script inclusion, postMessage, fragment identifier, etc
- SOP prevents cross-origin AJAX requests
  - Prevents attacker stealing anti-CSRF token by loading target page via AJAX
- Cross-Origin Resource Sharing (CORS) relaxes SOP for servers that opt in
  - Browser attaches `Origin=`*origin* header to cross-origin AJAX request
    - Upon redirection, `Origin` is set to null
  - If server accepts cross-domain requests from *origin*
    - It replies with header `Access-Control-Allow-Origin:` *`origin`* (or * for any origin)
    - Browser allows AJAX response to be received by script
  - If server does not care for CORS, response still reaches browser but is discarded

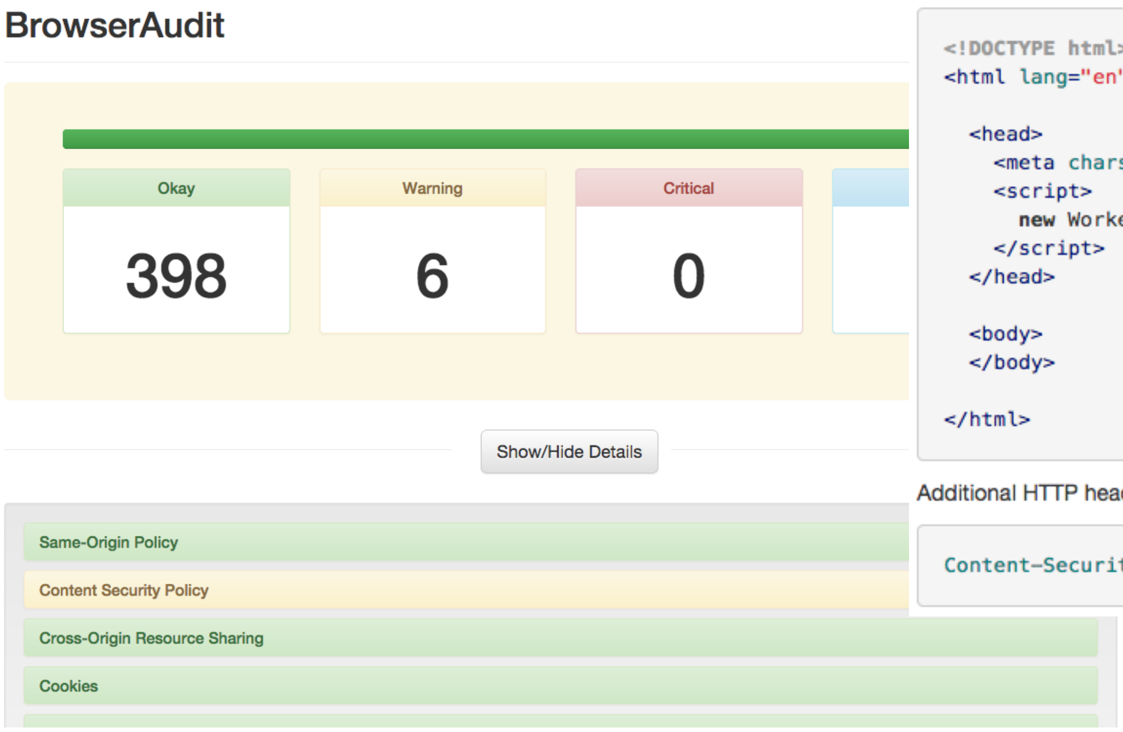

JavaScript
Browser
Server
xhr.send();
preflight request (if necessary)
preflight response (if necessary)
actual request
actual response
Fire onload() or onerror()

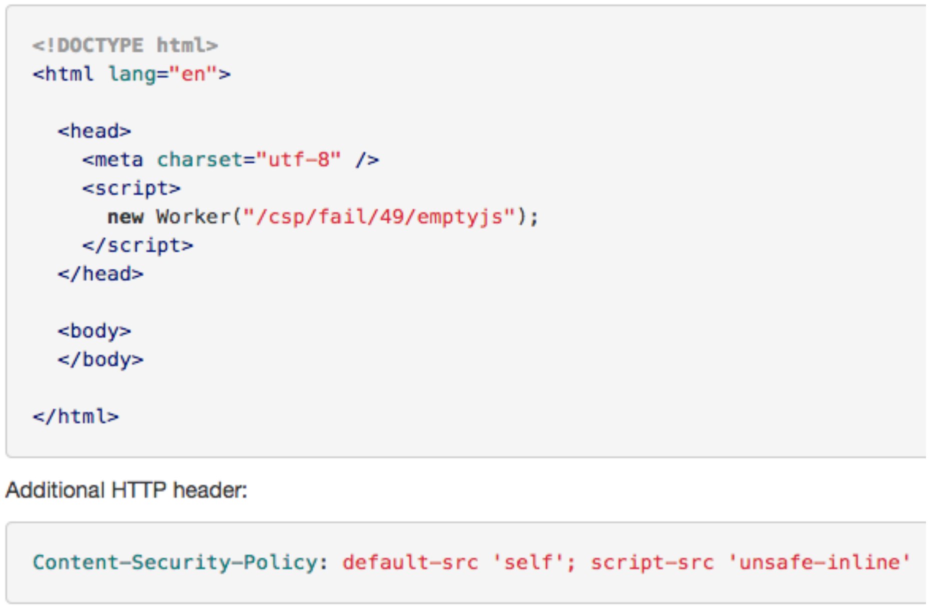
BrowserAudit

Imperial College London

[*Hothersall-Thomas, Maffeis, Novakovic:* ISSTA'15]

- Automated testing framework for SOP, CSP, CORS, HSTS
- Started as award-winning BEng individual project at Imperial
  - Charlie Hothersall-Thomas (Netcraft)
- Test if a policy provides the expected security behaviour
- User can inspect test source code to understand policy intent
- Discovered security issues in Firefox, Chrome, Blackberry





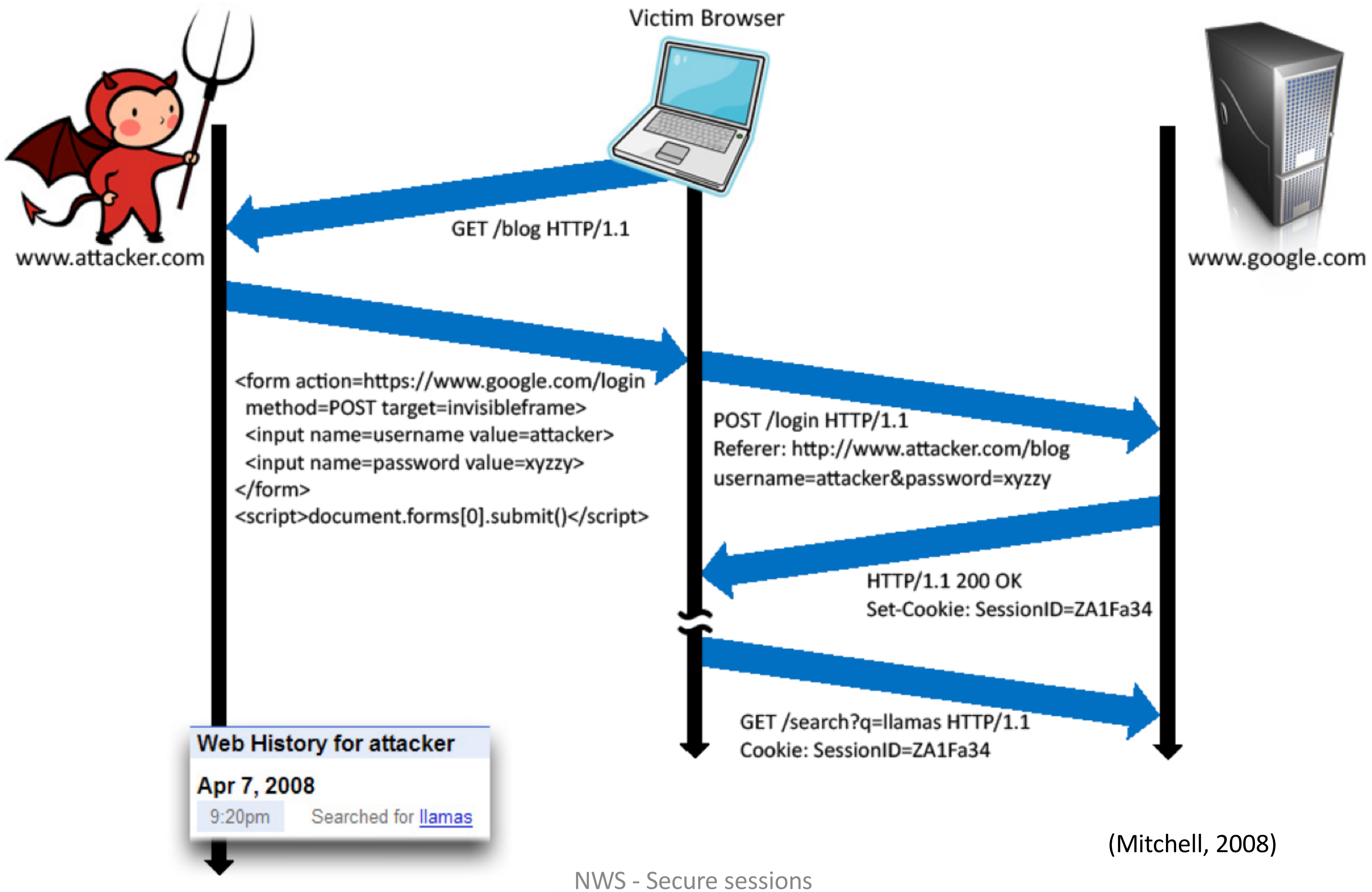
```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="utf-8" />
    <script>
      new Worker("/csp/fail/49/emptyjs");
    </script>
  </head>

  <body>
  </body>

</html>
```

Additional HTTP header:

```
Content-Security-Policy: default-src 'self'; script-src 'unsafe-inline'
```

Attacks: login CSRF

Imperial College London



```
<form action=https://www.google.com/login
  method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

(Mitchell, 2008)

Login CSRF mitigations

- Anti-CSRF token does not apply
  - Before login there is no session token to serve as 2nd-factor
- Validate `refer` or `origin` header of login request
  - Previous example:
    - POST request to `https://www.google.com/login`
    - With `referer` header `http://www.attacker.com/blog`
    - Very suspicious!
  - Only a partial mitigation
    - Sometimes `referer` and `origin` headers are stripped by network proxies, user preferences
    - See further reading
- Embed login form on a dedicated page
  - Served over HTTPS
  - From segregated domain that serves no other resources
  - Do not include 3-rd party scripts or iframes
  - This minimizes the risk of XSS, other mistakes

Secure sessions

1. Use HTTPS wherever possible: also before/after login
2. Segregate login in a secure domain
3. Change session token after login
4. Protect sensitive actions with anti-CSRF token cryptographically related to session token
   – Possibly also related to action itself
   – Or use SameSite cookies if compatible with web application deployment constraints
5. Use specific and short-lived tokens
   – If same token used more than once, MITM can launch replay attacks
   – The more specific the token, the harder to generate and maintain, but the better the protection
6. Check `referer` header where available
7. Ask for re-authentication for special actions
   – Transfer money to a new bank account
   – Change email or password
   – Delete account
8. After predetermined idle time, session should expire, or at least degrade to lower security
   – For example, read only access