Unraveling the Web: An Introduction to Penetration Testing

Chuanshu

May 4, 2025

Outline

Hello BSides Ballarat!

Me

- I have been a pentester for 3 years
- Occasionally I play capture the flag (CTF)
- I love puzzles

Penetration testing

- Offensive side of security
- Hired by companies to hack their systems before they are hacked by someone else
- Wide array of skills
  - Application testing (Web, desktop, mobile etc)
  - Network testing
  - Phishing/Social engineering campaigns
  - Physical access testing

Web application testing

- Many different kinds of web apps exist
- Look for common vulnerabilities
  - OWASP (Open Worldwide Application Security Project)
  - MITRE CWE (Common Weakness Enumeration)

Talk aims

- Explore a simple web attack: Cross site request forgery
- Overview of modern web security mechanisms
- Aim: To give a responsible introduction to web application hacking

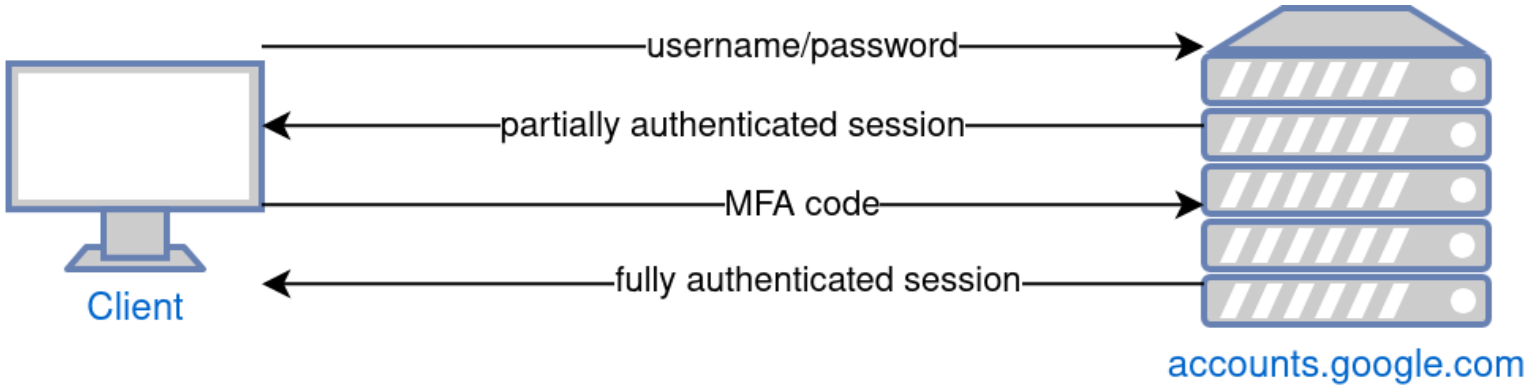
Web security concepts

- Authentication
- Browser features
  - Cookies
  - localStorage

Authentication

- Authentication is like the front door to an application
- Traditionally username/password
- Big push for Multi factor authentication (MFA)

Authentication



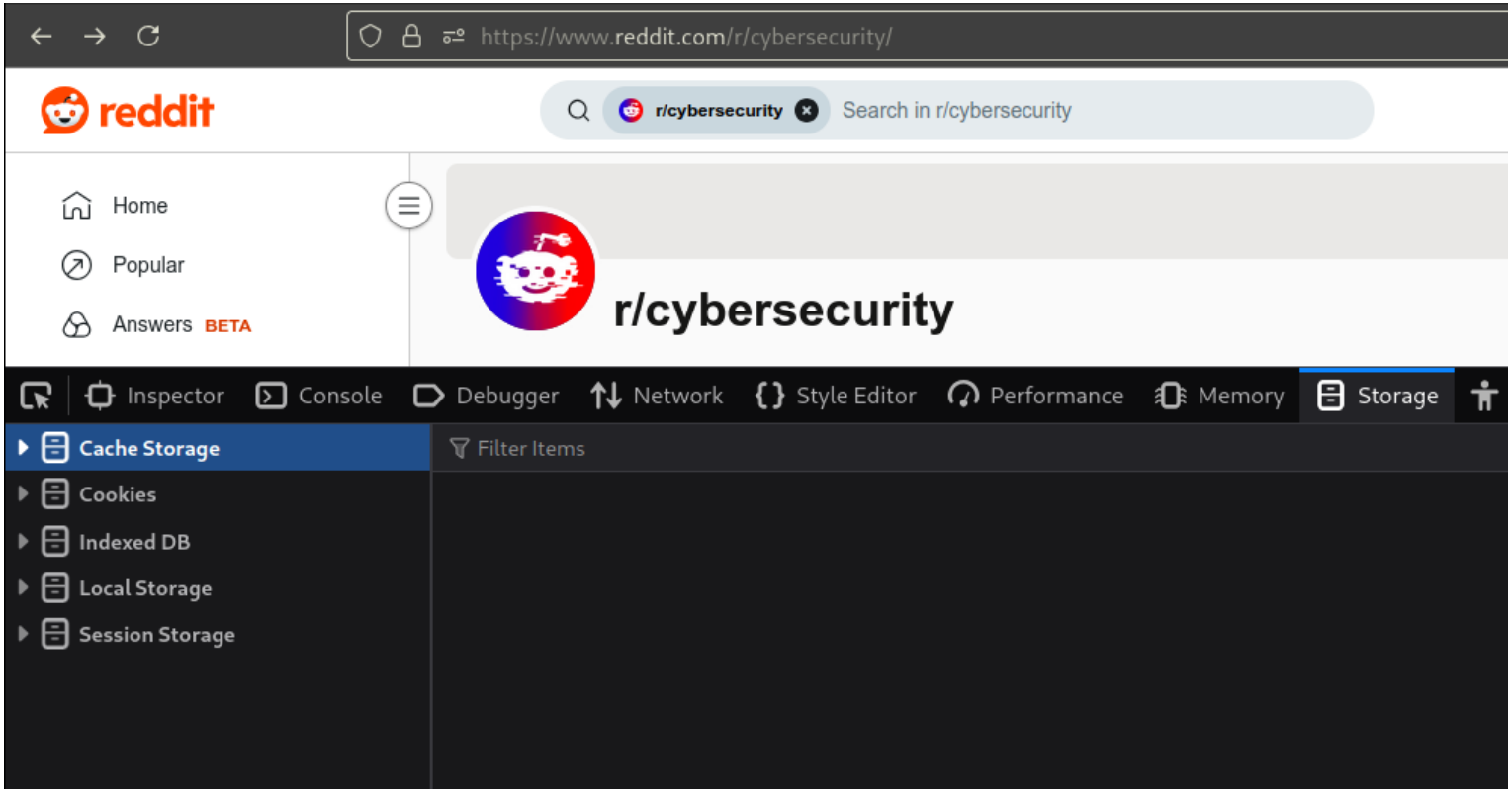
- Server returns information about authentication status
- Authentication information stored in browser

What's in a browser

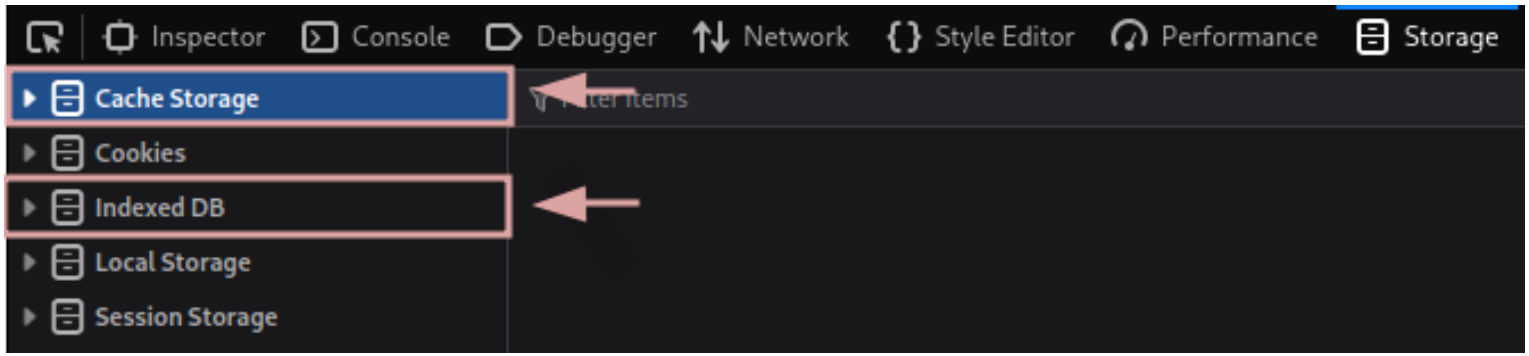
- We are particularly interested in where secrets are stored
- Browser storage has gotten much more complex over time to support complex app designs and requirements
- Browser storage is designed with security in mind [1]

[1] https://infosec.mozilla.org/guidelines/web_security

What's in a browser



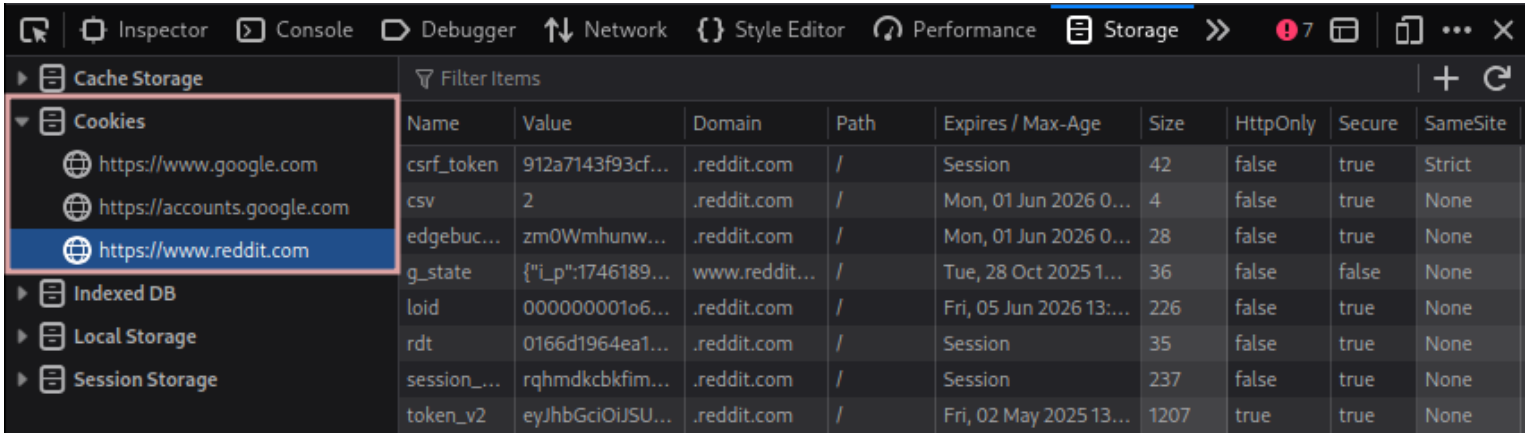
What's in a browser



- Cache storage and IndexedDB are for web workers [2]
- Wont be going through them in this talk, recommended reading if interested:
  - `https://portswigger.net/research/hijacking-service-workers-via-dom-clobbering`

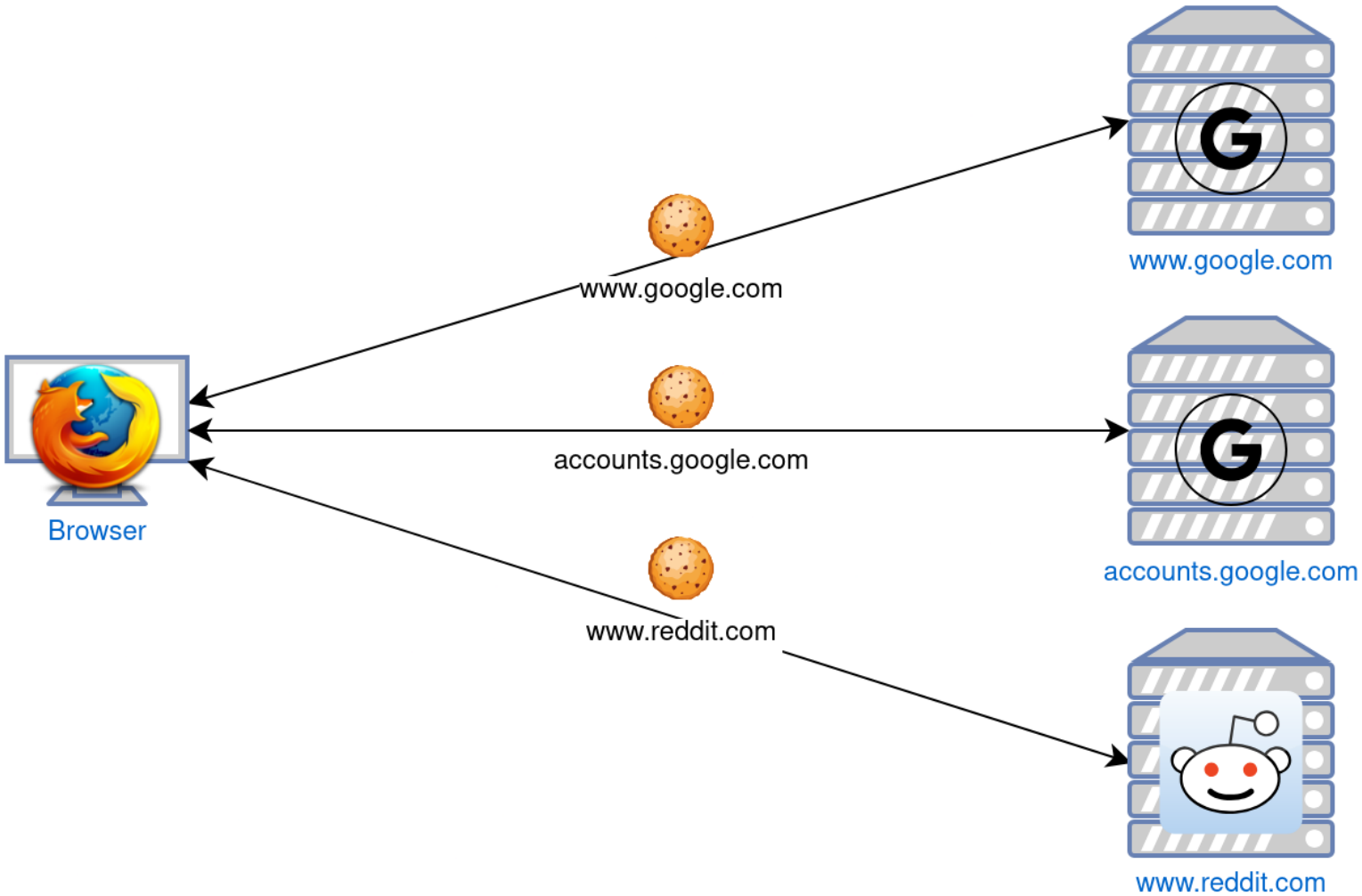
[2] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API`

Cookies

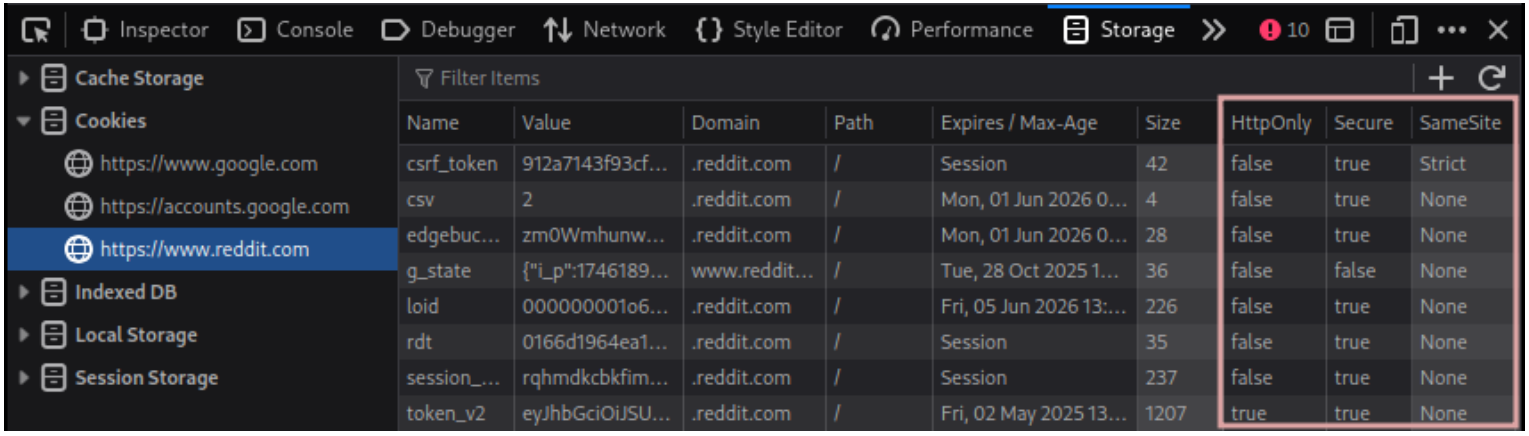


- Cookies were the original way for apps to store data in a user's browser
- Partitioned based on web origin, e.g. accounts.google.com and www.reddit.com are different origins
- Sent on each request to the relevant origin (almost always)

Cookies


www.google.com
www.google.com
accounts.google.com
accounts.google.com
Browser
www.reddit.com
www.reddit.com

Cookies



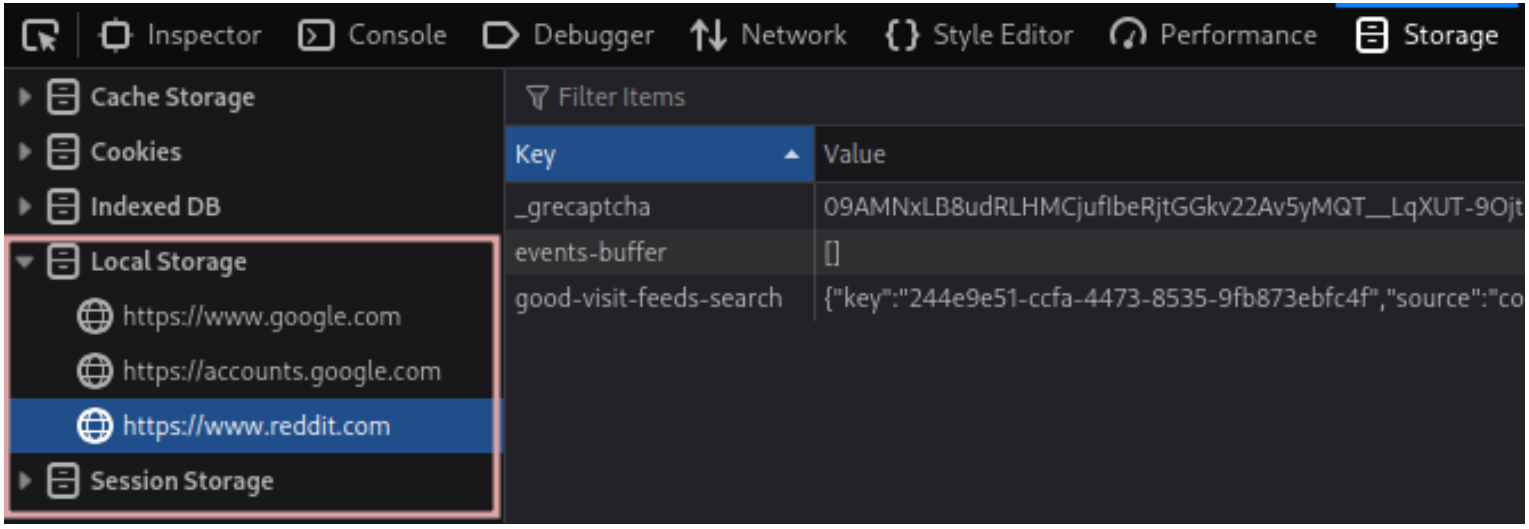
HTTPOnly Cannot be accessed by JavaScript

Secure Only transmitted over HTTPS connections

SameSite Whether cookies are sent over cross-origin requests

- `Strict`, `Lax` (default), and `None`

localStorage and sessionStorage

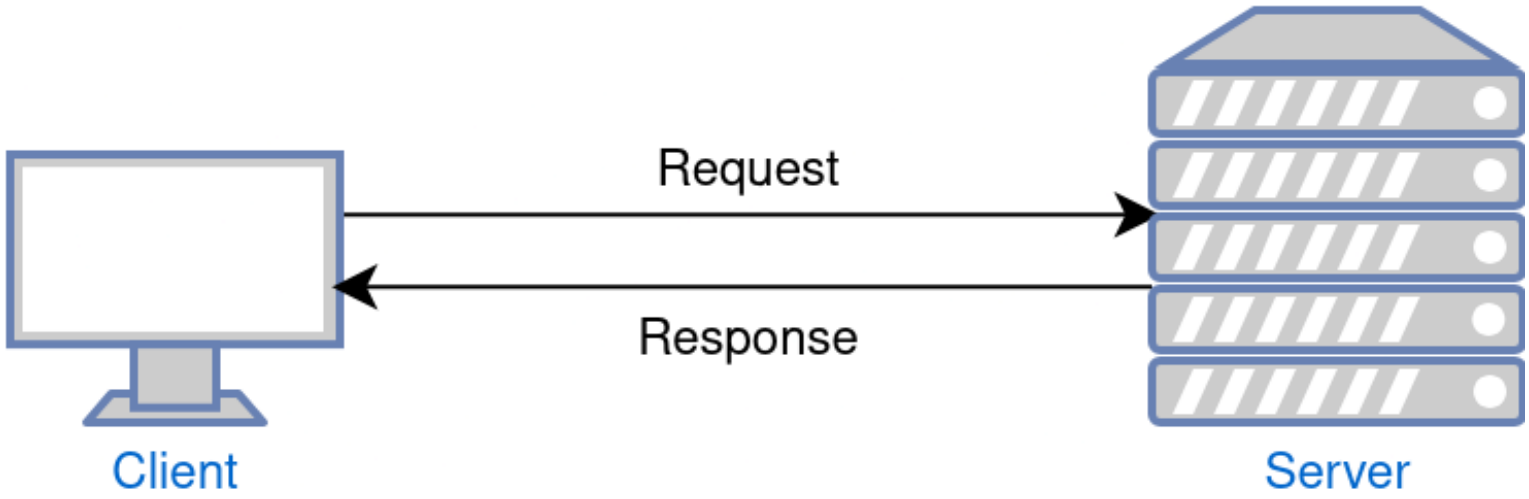


- Used to store much larger amounts of data that doesn't need to be sent to the server on each request
- localStorage persists across sessions, sessionStorage cleared when page is closed or reloaded
- Also partitioned by origin by the browser, always accessible by JavaScript

Web security concepts summary

- Authentication requires exchanging information such as username or passwords for a secret stored in browser
- Browser storage is separated by web origin
- Cookies are small bits of data sent with each request (most of the time)
- localStorage and sessionStorage are a newer storage mechanism

Cross Site Request Forgery



- CSRF is a client side attack
- Client-side attacks target the application running in another user's browser
- Server-side attacks target the application server or other backend components

Cross site request forgery

- A common goal is to leak information stored inside the target's browser, usually authentication secrets to gain control of that user's accounts
- Requires the target to load malicious content in their browser: "1-click" attacks

Cross site request forgery (CSRF)

- Rather than leaking secrets, the goal of CSRF attacks is to perform actions in the vulnerable web app as the target user
- If leaking client secrets is like stealing someone's keys, CSRF is like tricking someone inside the house into opening the door for you
- Scenario: The target clicks on a link sent to them by an attacker while logged in to the vulnerable application:
  - i.e. the secrets for that application are stored in the target's browser

Example 1: GET based CSRF

- Weakness: The web app `bank.com` uses a GET request to update user passwords

  `GET /change-password?new-password=<new password>`

- When the target clicks the link below, the browser automatically sends a GET request with the parameter `?new-password=hacked!1`.
- If the target is authenticated using cookies, this request will be authenticated

```
https://bank.com/change-password?new-password=hacked!1
```

Example 2: POST based CSRF

- Weakness: *SameSite* attribute of authentication cookie is `None`
- The snippet below could be included within a completely benign looking page or email
- Doesn't work if *SameSite* attribute is `Secure` or `Lax`

```
<form method="post" action="https://bank.com/change-password">
  <input type="hidden" name="new-password" value="hacked1!"/>
  <input type="submit" />
</form>
<script>document.forms[0].submit()</script>
```

Example 2: POST based CSRF (discussion)

- Browser protects you against POST based CSRF attacks as Cookies are set to *SameSite* by default
- Security guidelines should forbid using GET requests to make state changing operations

Interlude

- CSRF issues were rampant circa 2012
- Browsers now make it very difficult to make your app vulnerable to CSRF

Example 3: No cookies involved

- App used `Authorization` header to send user tokens
  - Tokens were stored in localStorage of `admin.app.com`
- Weakness: The app automatically makes authenticated requests using variables in the URL
- Browsing to `https://admin.app.com/users/1000` sends the request:

Request

```
GET /users/1000
Host: api.app.com
Authorization: <token>
```

Example 3: No cookies involved

- Browsing to `https://admin.app.com/users/foo` gave me a "User not found" error:
- Browsing to `https://admin.app.com/users/%252f` gave me a different error:

Request

```
GET /users/%2f
Host: api.app.com
Authorization: <token>
```

Response

```
404 Not Found
...

{
  "status": "404",
  "error": "Path not found",
  "path": "/users//"
}
```

Example 3: No cookies involved

- Checked API routes to see if there were any state changing GET requests
- There was one: `/admin/verify-user/1000`
- Use path traversal to navigate backwards: `/users/../admin/verify-user/1000`
  - Note: Directly accessing `admin.app.com/admin` wouldn't work as there was no automatic API call
  - Note: Directly accessing `api.app.com` wouldn't work as the user token was stored in localStorage on `admin.app.com`

Example 3: No cookies involved

- Malicious URL:
  - `https://admin.app.com/users/%252e%252e%252fadmin%252fverify-user%252f1000`

Request

```
GET /admin/verify-user/1000
Host: api.app.com
Authorization: <token>
```

Response

```
200 OK
...

{
  "message": "user verified"
}
```

Example 3: No cookies involved

- Possible to create a link that sent arbitrary authenticated GET requests on behalf of the user who clicks on the link
- Reshaped my idea of CSRF attacks
  - Look for automatic requests being sent → Sources
  - Look for APIs that could be consumed using these automatic requests → Sinks
- The idea of a POST sink was not unreasonable

Example 3: No cookies involved (discussion)

- Cause of the issue was not due to legacy systems but rather a redesign of their frontend
- Single page apps are increasing in popularity - these apps load a single HTML page and rely on JavaScript to render different components

Relevance to today

- Web apps are becoming increasingly client heavy
- Popularising of client-side protocols such as OAuth
  - Recommend reading my blog!
- Targeted and sophisticated phishing incidents occur very frequently in large companies

Conclusion

- Web technologies will continue to get more complex to support the needs of different application designs
- Complexity often creates security issues
- Secure by default, insecure by choice
- Problems arise when features designed for a certain purpose are used in unexpected ways

Recommended study

- Educational labs
  - PortSwigger labs (free)
  - Pentesterlab (free and paid option)
  - Hack The Box
- Wargames
  - OverTheWire
- CTFs
  - PicoCTF for practice
  - DownUnder CTF (local)
  - Happening all the time, find at ctftime.oeg
- Blogs & Articles
  - Infosec aggregator: talkback.sh

Questions?