# Broken Authentication and Session Management

Troy Hunt

troyhunt.com

pluralsight

hardcore developer training

## Outline

- How OWASP views the risk
- Performing an attack
- Understanding and configuring session persistence
- Using native authentication and membership features
- Configuring timeouts

# OWASP overview and risk rating

| Threat Agents |
|---|
| — |
| Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions. |

## Persisting state in a stateless protocol

- **HTTP is a stateless protocol**
  - Subsequent connections are entirely independent to previous ones
  - Uniquely – and securely – identifying the same user across multiple requests needs to be manually constructed
  - We do this by persisting a piece of data unique to the user across requests

## Session persistence in the URL

- **How it works:**
  - An ID unique to the session is generated
  - That ID is then returned in the URL
  - All subsequent requests by the browser include the ID in the URL

- **What’s wrong with this?**
  - URLs are often shared (social media, email)
  - URLs are also often logged (proxies, web server logs)
  - URLs are retrievable from browser history

## Session persistence in a cookie

- **How it works:**
  - An ID unique to the session is generated
  - That ID is then returned in a cookie
  - All subsequent requests by the browser send the cookie with it

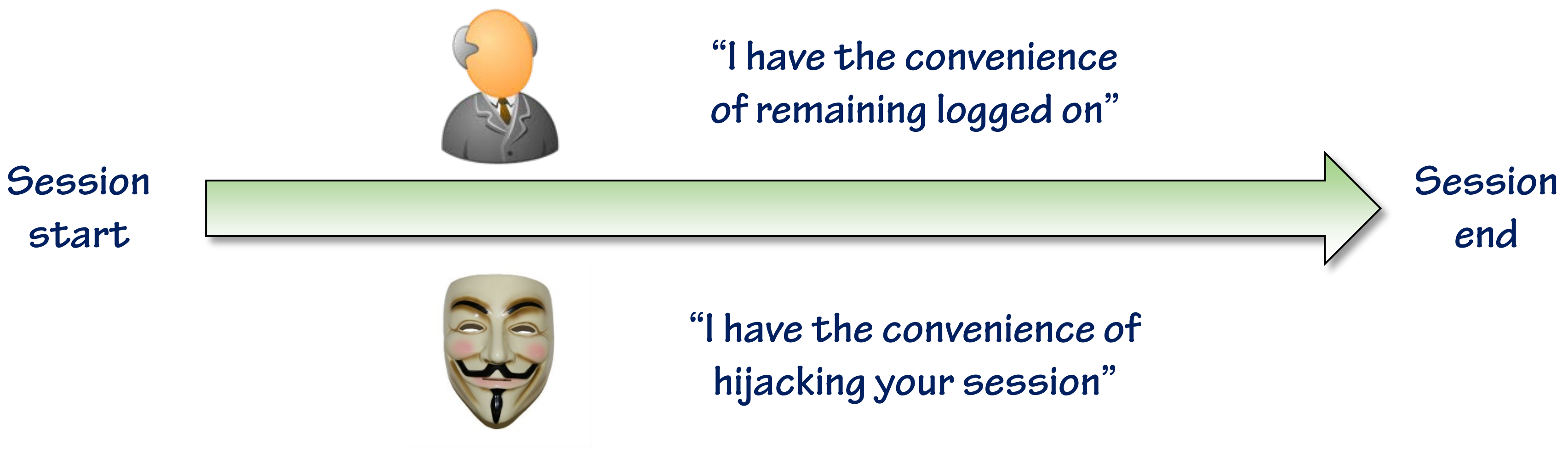
- **What's wrong with this?**
  - The browser needs cookies enabled

## Defining session persistence

| | |
|---|---|
| UseUri | Always use the URL regardless of device support |
| UseCookies (default) | Always use cookies regardless of device support |
| UseDeviceProfile | ASP.NET determines if cookies are *supported* in the browser and falls back to the URL if not |
| AutoDetect | ASP.NET determines if cookies are *enabled* in the browser and falls back to the URL if not |

## Session and forms timeout

- Timeouts control both the window of convenience and the window of risk



"I have the convenience of remaining logged on"

"I have the convenience of hijacking your session"

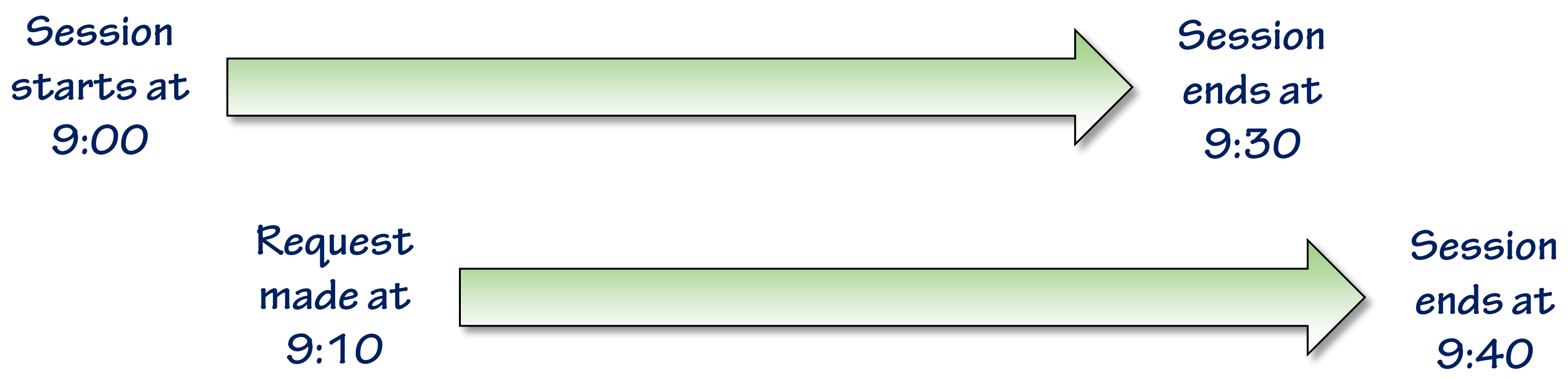
## Session and forms timeout

- The *session* timeout default to 20 minutes
- The *forms* timeout defaults to 30 minutes
  - But the Visual Studio templates set it to *two days!*

```
<forms loginUrl="~/Account/Login" timeout="2880" />
```

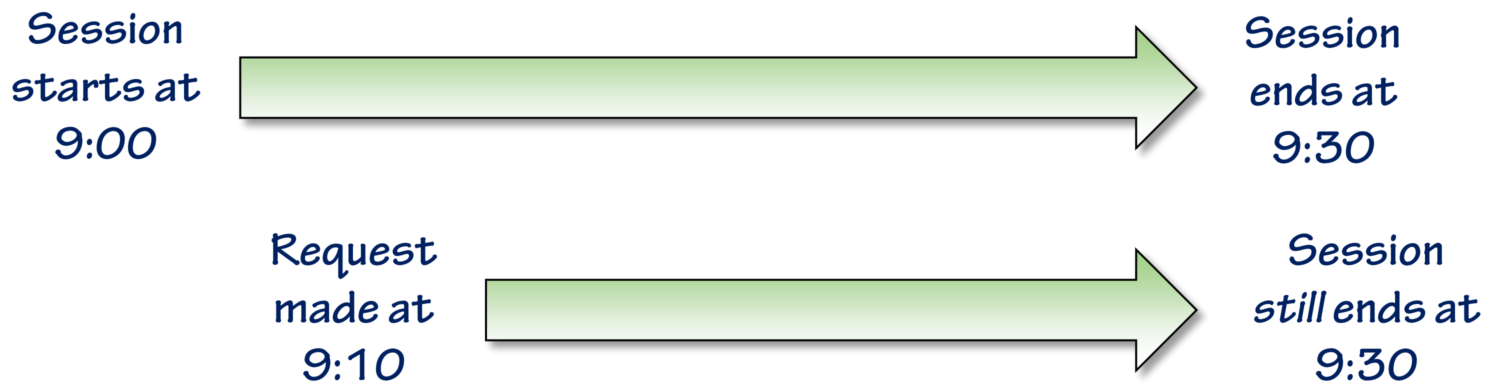
## Sliding forms timeout

- By default, forms timeout is *sliding*
- Imagine a 30 minute forms timeout



# Fixed forms timeout

- Now let's change this to a *fixed* timeout
- The duration is still 30 mins



## Configuring sliding expiration

- By default, the forms timeout is sliding
- It can be set to fixed by disabling sliding expiration

```
<forms slidingExpiration="false" />
```

## Other broken authentication patterns

- **Credentials should always be stored in a cryptographically secure way**
  - More on that in part 7 on insecure cryptographic storage
- **Implement robust minimum password criteria**
  - The membership provider makes this very easy
- **Never send a password by email**
  - Always implement a secure password reset process
- **Protect session IDs in cookies**
  - Implement robust protection against XSS risks
  - Don’t transmit sensitive data over an insecure connection

# Summary

- **Keep session IDs out of the URL (use the more secure cookie default)**
  - Don’t expect anything in the URL to be secure
- **Make use of the ASP.NET membership provider**
  - It abstracts away all the hard work of managing authentication
- **Customise your session and forms timeout**
  - Find the right balance between convenient and secure
- **If possible, disable sliding forms authentication expiration**
  - Consider the potential adverse impact on usability
- **Remember that broken authentication and session management is a broad risk**
  - Don’t forget the other areas of the Top 10 that can jeopardise this one