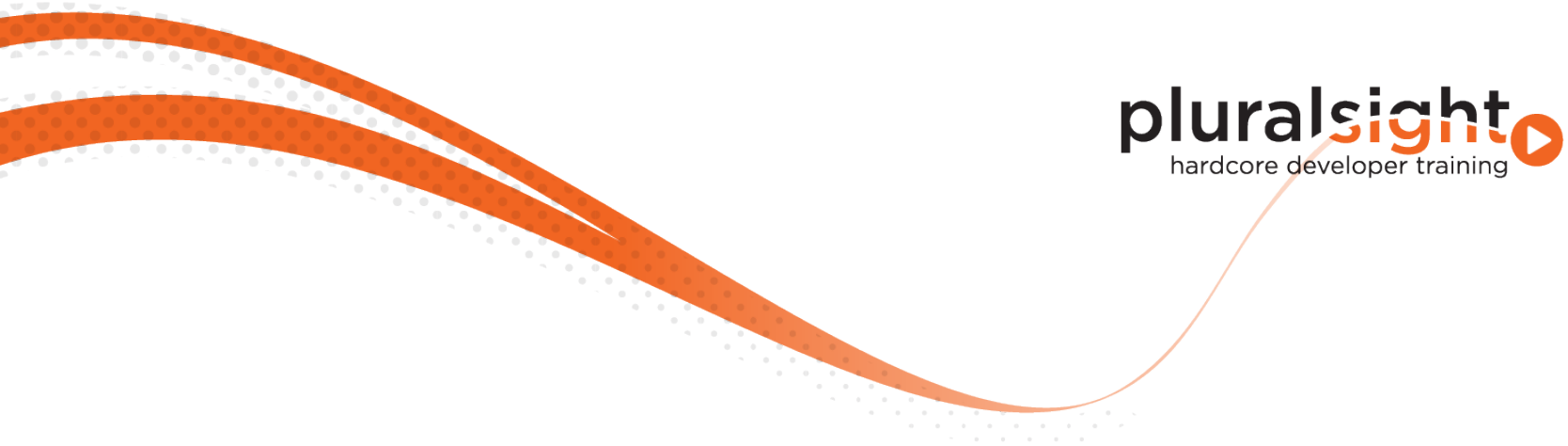
Cross Site Request Forgery (CSRF)

Troy Hunt
troyhunt.com



Outline

- How OWASP views the risk
- Performing an attack
- Understanding CSRF and anti-forgery tokens
- Protecting a vulnerable MVC application
- How web forms approaches CSRF mitigation
- CSRF fallacies and browser defences

OWASP overview and risk rating

Threat Agents
—
Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users access could do this.

What makes a CSRF attack possible?

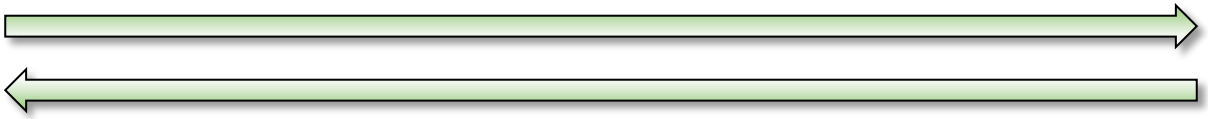
- **Authenticated sessions are persisted via cookies**
  - The cookie is sent with every request to the domain
- **The attacking site recreates a legitimately formed request to the target site**
  - Although the request has a malicious payload (query string parameters or post data)
- **The victim's browser is tricked into issuing the request**
  - For all intents and purposes, the target website views it as a legitimate request

Understanding anti-forgery tokens

- **CSRF attacks work because they're predictable**
  - The attack is merely reconstructing a request adhering to the same structure as a legitimate one (path and parameters)
- **To mitigate this risk, we can add randomness via a CSRF token**
- **A token is a random string known to both the legitimate page where the form is and to the browser via a cookie**

Anti-forgery tokens in action

Page with a form is requested



Resultant page contains a token in a hidden field and *also* one in a cookie

Browser sends back both the hidden form token and the one in the cookie

Server ensures they match and rejects the request if not

CSRF fallacies

- **Implement referrer checking to restrict cross-domain requests**
  - Can help, but doesn't address risks introduced by XSS
- **Disable HTTP GET on at-risk pages**
  - Helps mitigate some attack vectors (such as image source requests) but the attacker can still construct POST requests
- **Validate that the IP address posting the data is the same as the one loading the page**
  - Pointless as it's still the victim's browser in both cases

Native browser defences

- **The CSRF risk may be exploited by using Cross-Origin Resource Sharing or CORS**
  - Different browsers provide varying levels of defence against unintended CORS requests
- **As with other native browser defences (i.e. XSS), they should never be relied on**
  - The key always comes back to writing secure code which for CSRF ultimately means anti-forgery tokens

Summary

- **CSRF is made possible when a legitimate request is reconstructed into one with malicious intent**
- **The authenticated state of the victim is abused and the browser tricked into issuing the request**
- **The only real mitigation is anti-forgery tokens**
  - Easy in MVC, messier in web forms
- **Because the request is issued by the victim's browser, it appears legitimate thus difficult to implement other defences around**
- **Like with XSS, browsers offer some defences**
  - Also like XSS, don't rely on them!