Dissecting CSRF Attacks & Defenses

Mike Shema
Vaagn Toukharian
Cross Site Request Forgery

• The confused, session-riding deputy

• Pros & cons of current countermeasures

• Improving verification of CSRF tokens -- DEMO

• Improving defenses -- DEMO & SPECS!
By Your Command

- Cross-origin requests are a core part of how the web works.
- Effective CSRF only cares about generating a request that affects a server-side context.

```html
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="refresh" content="0; url=https://another.origin/CSRF">
  <link ref="prefetch" href="https://another.origin/CSRF">
</head>
<body>
  <img src="https://another.origin/CSRF" alt="">
  <iframe sandbox src="https://another.origin/CSRF"></iframe>
</body>
</html>
```
• Fundamentally, we want to distinguish between a user-intended action and a browser-initiated one.
  • Cross-origin requests that assume the victim’s authorization are the problem (i.e. session riding)

• HTML thrives on aggregating content from different Origins -- there’s no reason to change this.
Forging Ahead

• Creation
  • SOP restricts reading the response from a cross-origin request, not making one
  • Cross Origin Resource Sharing makes aggregation more flexible -- and has positive implications for blocking CSRF.

• Counterfeit
  • Predictable name/value pairs
  • Valid, invalid, stripped Referer, Origin headers
Castles Made of Sand

• Tie the request to the user’s session -- authorization vs. authentication.

• Add a secret (e.g. entropy) to make it harder to counterfeit
  • Double submit cookie
  • Anti-CSRF token (nonce)
Secrets & Entropy

- PRNG
- $\text{hash(}\text{hash(}\text{hash(}...(\text{PRNG})...)))$
- HMAC-SHA256(\text{PRNG}, \text{secret})
  - HMAC-MD5
  - HMAC-SHA512
HMAC

• Requires a strong secret
• Something other than the default value
  • “keyboard cat”
• Something outside a dictionary
  • 1
  • 123
  • secret
  • Shad0wfax
• Distributed, collaborative secrets
  
  • http://www.phenoelit.org/blog/archives/2012/12/21/let_me_github_that_for_you/
  
  • http://nakedsecurity.sophos.com/2013/01/25/do-programmers-understand-private/
Entropic Horror

• BH2012 -- PRNG: Pwning Random Number Generators
• sjcl.random
• openssl rand 32 -hex
CSRF Cloaks Bad Design

• POST/GET method ignorance
• Password change mechanisms that don’t require current password
• Missing barriers that rely on authentication to perform actions.
  • e.g. check-out and shipping to known vs. new address
• Loose coupling of authentication, authorization, and session.
Mobile Apps

• Recreating vulns from first principles
  • Using HTTP instead of HTTPS
  • Not verifying HTTPS certs
  • But at least the apps are signed...

• More areas to explore
  • Not a browser, but making HTTP requests
  • CSRF potential of malevolent ad banners
Detection

• Pattern-based detection of token names
  • Security by regex-icity
  • Checks for presence, not effectiveness

• Active test
  • “Cookie Swap” between user session contexts
  • Determine enforcement, not predictability
DEMO
Cross Origin Resource Sharing

- Control the forgery (i.e. creation) of “non-simple”, cross-origin requests
  - X-CSRF: 1
  - XCSRF /foo HTTP/1.1
Rely on SOP & HTML5

- Guarantees same Origin (or allowed cross-Origin)
  - But only for “non-simple” XHR requests
  - Must start inspecting the Origin header

- Limitations
  - Must be part of app’s design and implementation
  - Breaks “simple” cross-origin requests
Crosstown Traffic

- HTML injection, cross-site scripting
  - It’s executing in Same Origin
  - CSRF countermeasures are intended to prevent cross-origin attacks
  - Start using Content Security Policy

- DNS, cache poisoning, sniffing, ...
  - Start using HSTS
  - Where did DNSSEC go?
Background Radiation of Insecurity

20 months starting November 2011
Speaking of CSP

```html
<!doctype html>
<html>
<head>
<meta http-equiv="X-WebKit-CSP" content="img-src 'none'; report-uri 'https://csrf.target/page?a=1&b=2&c=3'">
</head>
<body>
<img alt="" src="whatever">
</body>
</html>
```
Partial POST Request Forgery

POST /page?a=1&b=2&c=3 HTTP/1.1
Host: csrf.target
User-Agent: Mozilla/5.0 ... 
Content-Length: 116
Accept: */*
Origin: null
Content-Type: application/x-www-form-urlencoded
Referer: http://web.site/HWA/ch3/csrf.html
Cookie: sessid=12345
Connection: keep-alive

document-url=http%3A%2F%2Fcsrf.target%2FHWA%2Fch3%2Fcsrf.html&violated-directive=default-src+%27none%27
AND THEY HAVE A PLAN.
Security of Sessions

• Focus on the abuse of session context
  • Session-riding, confused deputy

• Control when cookies accompany requests initiated from a cross-origin resource
  • Similar to CORS enforcement of “non-simple” requests
  • Isolate the user’s session context
Simplicity of Settings

• Syntax like CSP, behavior like CORS
  • Simple behavior with fewer chances of mistakes
  • Leverage pre-flight

• Don’t require changes to application code
  • Add headers via WAF
  • Provide more flexibility by opt-in to exceptions
Should Often Succeed

- Don’t break the web, ease adoption
  - Ad banners
  - “first visit”, blank browsing context
  - Deal with domains & subdomains vs. Origins

- Browsers have to support it
  - Old, unpatched browsers forsaken to the demons of insecurity anyway
Some Ordinary Syntax

• On the web application, define a policy:

Set-Cookie: cookieName=...
Content-Security-Policy:

sos-apply=cookieName; ‘self’
sos-apply=cookieName; ‘any’
sos-apply=cookieName; ‘isolate’
sos-apply=*=; ‘self’
Policies

- **self** -- trigger pre-flight, cookie included only from same origin unless given exception
- **any** -- trigger pre-flight, cookie included unless given exception
- **isolate** -- no pre-flight, no exceptions. Cookie only included from same Origin.

- (?) sos-remove=cookieName to remove policy
Some Ordinary Syntax

- If a cookie has a policy (or no policy), and a request is generated by a resource from the same Origin.
  - ...work like the web works today.
- If a cookie has a policy of ‘isolate’, and a request is generated by a cross-origin resource.
  - ...never include the cookie.
- If a cookie has a policy of ‘any’ or ‘self’, and a request is generated by a cross-origin resource.
  - ...make a pre-flight check
Why Pre-Flight?

- Cookies apply site-wide (including subdomains!), without granularity of resources.
  - The /path attribute is broken

- An SOS policy instructs the browser for default handling of a cookie.

- A particular resource can declare an exception by responding to the pre-flight.
Pre-Flight Request

- (prereq) A policy of ‘any’ or ‘self’
- (prereq) Cross-origin resource initiates request
- Browser makes CORS-like request:

```
OPTIONS http://web.site/resource?a=1&b=2 HTTP/1.1
Host: web.site
User-Agent: ...
Origin: http://evil.site
Access-Control-SOS: cookiename cookiename2
Connection: keep-alive
Content-Length: 0
```
• Web app receives a pre-flight request.
• Supply an expires value so the browser can cache the response.
• ...if a policy should be enforced for the specific resource:

HTTP 200 OK
Access-Control-SOS-reply: ‘allow’ | ‘deny’; expires=seconds
Pre-Flight Response

• ...if the resource is not exceptional, browser follows established policy
  • ‘any’ would include the cookie for cross-origin
  • ‘self’ would exclude the cookie for cross-origin

• Benefits
  • Web app can enforce per resource, per cookie
  • Sees the Origin header
  • Expiration eases performance with caching
Two Sets

• Policy applies to cookies for all resources (entire Origin)
• Policy can be adjusted by a resource
• Pre-flight response shouldn’t leak information about cookies for which it has a policy
  • If the client can’t ask for the right cookie, then no response.
  • Respond with ‘deny’ if the cookie doesn’t exist
Remember

- Browser tracks...
  - Cookies for which a policy has been applied.
  - Resources that respond to cross-origin requests with exceptions to the policy.
  - Cookies and destination origin, source origin doesn’t matter
- Web App
  - Applies a policy at each Set-Cookie
  - Applies a policy at a bottleneck
Goals

- Ease adoption
  - Familiar syntax
  - Small command set

- Acknowledge performance
  - Cache pre-flight responses
  - Only track “all other origins” to origin, not pairs of origins
The “WordPress Problem”

• Strong anti-CSRF token is present in WordPress trunk

• WP plugins keep forgetting to use it
  • ../wp-admin/admin.php?page=...

• Must continually protect every new action

• ...or protect the /wp-admin/ directory
  • sos-apply=cookieName; ‘self’
Mitigate Social Engineering

• Should prevent situations where user is tricked onto clicking a link/submitting a form on attacker’s page (i.e. different origin) that submits to targeted origin

• Use X-Frame-Options to deal with clickjacking
If 6 Was 9

- No secrets, no entropy
  - Easier on embedded devices, fewer mistakes
- Enforcement by origin
  - Exception-based for flexibility
  - Shift state tracking from server to browser
- Pre-flight can be handled by WAF
- ‘isolate’ and expire deal with overhead of pre-flight
  - (Which is only for cross-origin anyway)
When Old Becomes New

• Update browsers
  • Still have to support legacy, although the window to the past is shrinking
  • People still use old browsers for good reasons, TorBrowser using FireFox ESR

• Fix frameworks
  • Use cryptographically secure PRNG
  • Don’t reuse example passphrases
  • Use XHR brokering with custom headers
  • Separate authentication and authorization
Summary

• Use HSTS

• Use CORS (i.e. “non-simple” requests)

• Send an SOS

SIX: ALL OF THIS HAS HAPPENED BEFORE.
BALTAR: BUT THE QUESTION REMAINS, DOES ALL OF THIS HAVE TO HAPPEN AGAIN?
Thank You!

- DefCon HTTP Time Bandit
  - Friday 2:30pm, Track 2

- http://deadliestwebattacks.com
References

• https://media.blackhat.com/bh-us-12/Briefings/Argyros/BH_US_12_Argyros_PRNG_WP.pdf