Robust Defenses for Cross-Site Request Forgery

Tsampanaki Nikoleta
Lilitsis Prodromos
Gigis Petros

Paper Authors: Adam Barth, Collin Jackson, John C. Mitchell
Outline

• What is CSRF attack?
• What is a login CSRF attack?
• What’s the existing CSRF defenses?
• What’s authors defense proposal?
• What’s the vulnerabilities of Session Initialization?
Outline

• What is CSRF attack?
• What is a login CSRF attack?
• What’s the existing CSRF defenses?
• What’s authors defense proposal?
• What’s the vulnerabilities of Session Initialization?
What is CSRF attack?

Cross-Site Request Forgery (CSRF) attack

- A malicious site instructs a victim’s browser to send a request to an honest site
- Leveraging the victim’s network connectivity and browser’s state, such as cookies, to disrupt the integrity of the victim’s session with the honest site
CSRF Defined

- Attacker take advantage of user
  - Network Connectivity
  - Read Browser State
  - Write Browser State

- Different attacker types
  - Forum Poster
  - Web Attacker
  - Network Attacker
Cross-Site Request Forgery

```
<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

HTTP/1.1 200 OK
Transfer complete!
```
Real attack using CSRF

Hacked Gmail accounts and injected forwarding mail filter
Outline

• What is CSRF attack?
• What is a login CSRF attack?
• What’s the existing CSRF defenses?
• What’s authors defense proposal?
• What’s the vulnerabilities of Session Initialization?
Login CSRF attack

www.attacker.com

GET /blog HTTP/1.1

<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

www.google.com

Search History
Paypal
iGoogle

Web History for attacker
Apr 7, 2008
9:20pm  Searched for llamas
Outline

• What is CSRF attack?

• What is a login CSRF attack?

• What’s the existing CSRF defenses?

• What’s authors defense proposal?

• What’s the vulnerabilities of Session Initialization?
What’s the existing CSRF defenses?

- **Secret Validation Token**
  used to determine whether the request came from an authorized source.

- **The Referrer header**
  indicates which URL initiated the request.

- **Custom HTTP header**
  using XMLHttpRequest
Secret Validation Token - Designs (1/2)

Session identifier

- Use the user’s session identifier as the secret validation token
- **Disadvantage:** Users may share the contents of WebPages that contain session identifiers to third parties

Session-Independent Nonce

- Server generates a random nonce and store it as a cookie when the user first visits the site.
- **Disadvantage:** An active network attacker can overwrite the session independent nonce with his own CSRF token value even if HTTPS is present (Cookie Overwriting attack).
Session-Dependent Nonce

- Store state on the server that bind the user’s CSRF token value to the user’s session identifier.
- **Disadvantage:** Site must maintain a large state table in order to validate the tokens

**HMAC of Session Identifier**

- Instead of using server-side state, cryptography is used to bind the CSRF token and the session identifier.
- All site servers share the HMAC key and each server can validate that the CSRF token is correctly bound to the session identifier.

*HMAC = Hash Message Authentication Code*
The Referrer header

When the browser issues an HTTP request it includes a Referrer header that indicates which URL initialized the request.

**Referrer disadvantages:**
- is usually suppressed due to privacy information leaking.
- can be spoofed due to browser bugs.

**Referrer validation as a CSRF defense**
- In lenient Referrer validation, the site blocks requests whose Referrer header has an incorrect value. If a request lacks the header, the site accepts the request.
  - Disadvantage: Referrer is suppressed for request issued from Ftp and data URLs.
- In strict Referrer validation, the site blocks requests that lack a Referrer header.
Experiment (1/4)

Design

- We used two advertisement networks from 5 April 2008 to 8 April 2008
- 283,945 advertisement impressions from 163,767 unique IP address
- Two servers with two domain names were used to host the advertisement
- Advertisement generates a unique id and randomly selects the primary server
- GET and POST requests both over HTTP and HTTPS generated by primary server
- Requests are generated by submitting forms, requesting images, and issuing XMLHttpRequests
- Same-domain requests to the primary server and cross-domain requests to the secondary server were generated by the advertisement
- Servers logged request parameters (Sids, Referer, User-Agent, document.referrer, etc)
- Did not log the client’s IP address, instead logged the HMAC of client’s IP address
Experiment (2/4)

- **Results**

![Bar chart showing suppression rates for different request types]

- **Discussion**
  - The Referrer header is suppressed more often for cross domain requests over http.
  - The Referrer header is suppressed more often for HTTP requests than for HTTPS requests.
Experiment (3/4)

- Browsers that suppress the Referrer header also suppress the document. referrer value
- PlayStation 3 browser does not support document. referrer
- Opera blocks cross-site document. referrer for HTTPS
- Bug in Firefox 1.0 and 1.5 do not send referrer for HTML HTTP request
Conclusion:

- Strict Referrer can be used as CSRF defense for HTTPS
- HTTP: cannot afford blocking requests that lack Referrer header
  - cease to be compatible with 3-11% of users
- Strict Referrer validation suited for login CSRF defense because they are issued over HTTPS
  - 0.05-0.22% of browsers suppress the header over https
Custom HTTP Headers

Browser prevents sites from sending custom HTTP headers to another site but allows sites to send custom HTTP headers to themselves using XMLHttpRequest.

To use custom headers as a CSRF defense a site must

- Issue all state-modifying requests using XMLHttpRequest
- Attach a custom header (e.g. X-Requested-By)
- Reject all state-modifying requests that are not accompanied with the header
Outline

• What is CSRF attack?
• What is a login CSRF attack?
• What’s the existing CSRF defenses?
• What’s authors defense proposal?
• What’s the vulnerabilities of Session Initialization?
Proposal: Origin header

**Privacy**
- Origin header improves on a Referer header
- Origin header includes *only the information required* to identify the principal that initiated the request (port, host, scheme)
- Origin header doesn’t contain the path or query portions of the URL
- Origin header is sent only for POST requests
- Referer header is sent for all requests

**Server Behavior**
- All *state-modifying requests*, including login requests, must be sent using the *POST method*
- Server must reject any requests whose Origin header contains an undesired value (indicated from another site/attacker) or null
Proposal: Origin header

Security Analysis

- **Rollback and Suppression**: A supporting browser will always include the Origin header when making POST requests, sites can detect that a request was initiated by a supporting browser by observing the presence of the header.

- **DNS Rebinding**: Sites that rely only on network connectivity for authentication, could complementary validate the Host header. It applies to all CSRF defenses.

- **Plug-ins**: If a site opts into cross-site HTTP requests an attacker can use Flash Player to set the Origin header in cross-site requests. Sites should not opt into cross-site HTTP requests from untrusted origins.
Proposal: Origin header

Adoption
- Origin Header improves and unifies other proposals and has been adopted by several working groups.

Implementation
- Browser side: WebKit, Safari, Firefox
- Server side: ModSecurity, Apache
Outline

• What is CSRF attack?

• What is a login CSRF attack?

• What’s the existing CSRF defenses?

• What’s authors defense proposal?

• What’s the vulnerabilities of Session Initialization?
Session Initialization

- Login CSRF is an example of vulnerability in session initialization

Authenticated as User
- The attacker can force the site to use a predictable session identifier for a new session. Then user supplies their authentication credentials to the honest site (predictable session identifier)

Authenticated as Attacker
- Attacker cause the honest site to begin a new session with the user’s browser but force the session to be associated with the attacker’s authorization (e.g. Login CSRF, PayPal)

Two common approaches to achieve an attack on session initialization
- HTTP Requests and Cookie Overwriting
HTTP Requests (1/2)

- **OpenID**: includes a self-signed nonce to protect against reply attacks but doesn’t suggest a mechanism to bind the OpenID session to the user’s browser.

- 1. Web attacker visits the Relying Party (Blogger) and begins the authentication process with the Identity Provider (Yahoo!)
- 2. Identity Provider redirects the attacker’s browser to the “return to” URL of the Relying Party
- 3. attacker directs the user’s browser to the return to URL
- 4. The Relying Party completes the OpenID protocol and stores a session cookie in the user’s browser
- 5. The user is now logged in as the attacker

**Defense**: relying party should generate a fresh nonce at the start of the protocol, stored in user’s browser.
HTTP Requests (2/2)

- **PHP Cookieless Authentication:** stores the user’s session identifier in a query parameter

- 1. The web attacker logs into the honest web site
- 2. The web attacker redirects the user’s browser to the URL currently displayed in the attacker’s location bar
- 3. Because this URL contains the attacker’s session identifier, the user is now logged in as the attacker

**Defense:**

Site could maintain a long-lived frame that contains the session identifier token. (stores the session identifier in memory)
Cookie Overwriting

**Vulnerability:** An active network attacker can supply a Set-Cookie header over a HTTP connection to the same host name as the site and install either a **Secure** or a **non-Secure cookie** of the **same name**

**Defense:** “Cookie-Integrity header" in HTTP requests, identifies the cookies that were set using HTTPS.
Questions?