A Server- and Browser-Transparent CSRF Defense for Web 2.0 Applications

Riccardo Pelizzi
System Security Lab
Department of Computer Science
Stony Brook University
December 8, 2011
Outline

• CSRF Attacks
Outline

• CSRF Attacks
• CSRF Defenses
Outline

- CSRF Attacks
- CSRF Defenses
- jCSRF
Outline

- CSRF Attacks
- CSRF Defenses
- jCSRF
- Conclusions and Q&A
CSRF Attacks

Victim
Please visit evil.com

Bank
Transfer money to attacker

POST bank.com - data + cookies

Attacker
Victim
Browser

jCSRF: A CSRF Defense for Web 2.0 Applications
CSRF Attacks

Attacker

Please visit evil.com

Victim

Click on link

Browser

POST bank.com – data + cookies

Bank

Transfer money to attacker
CSRF Attacks

Attacker

Victim

Please visit evil.com

Click on link

Browser

POST bank.com – data + cookies

Bank

Victim

Transfer money to attacker

3 / 18 Riccardo Pelizzi
CSRF Attacks

Victim
Please visit evil.com

Browser
Click on link
POST bank.com – data + cookies

Bank
Transfer money to attacker
CSRF Attacks (2)

Attacker

Victim

Please visit evil.com

Click on link

Browser

Malicious Server

Bank Server

Bank Database

Automatically submit form

GET evil.com

Send response

POST bank.com – data + cookies

Transfer money
Victim
Please visit evil.com
Click on link
Malicious Server
Bank Server
Bank Database
Attacker
Browser
GET evil.com
Send response

Automatically submit form

POST bank.com – data + cookies
Transfer money
CSRF Attacks (2)

Attacker

Victim

Browser

Malicious Server

Bank Server

Bank Database

Please visit evil.com

Click on link

GET evil.com

Send response

Automatically submit form

Transfer money
CSRF Attacks (2)
CSRF Attacks (2)

Victim

Please visit evil.com

Click on link

GET evil.com

Send response

Automatically submit form

POST bank.com – data + cookies

Transfer money

Attacker

Browser

Malicious Server

Bank Server

Bank Database

4 / 18 Riccardo Pelizzi

jCSRF: A CSRF Defense for Web 2.0 Applications
CSRF Defenses

- Harder: Must recognize which requests are intended.
  - Easier: allow requests from trusted origins.

Two techniques:
- Use explicit origin information.
  ```html
  Referrer : http://attacker.com/csrf.html
  ```
- Require source pages to (manually) insert a secret token in requests.
  ```html
  <form action="transfer.php" method="POST">
    <input type="hidden" name="CSRF-token" value="gdFGf4sjhf4FGFA-df4">
  ...
  </form>
  ```
CSRF Defenses

• Harder: Must recognize which requests are intended.
  • Easier: allow requests from trusted origins.

• Two techniques:
  • Use explicit origin information.
    Referrer: http://attacker.com/csrf.html
  • Require source pages to (manually) insert a secret token in requests.
    <form action="transfer.php" method="POST">
      <input type="hidden" name="CSRF-token" value="gdFGf4sjhf4FGFA-df4">
      ...
    </form>
Key Benefits of Our Approach

• How does our solution improve the state of the art?
Key Benefits of Our Approach

• How does our solution improve the state of the art?
  • Handles requests dynamically generated on the browser from JavaScript code.
Key Benefits of Our Approach

• How does our solution improve the state of the art?
  • Handles requests dynamically generated on the browser from JavaScript code.
  • Can authenticate cross-origin requests.
Key Benefits of Our Approach

• How does our solution improve the state of the art?
  • Handles requests dynamically generated on the browser from JavaScript code.
  • Can authenticate cross-origin requests.
  • Is compatible with all browsers and transparent to web applications and languages.
Overview

User
Visit safe.com

Browser
GET

jCSRF Proxy
GET
Response

Server
PO ST

jCSRF: A CSRF Defense for Web 2.0 Applications
Overview

User

Browser

jCSRF Proxy

Server

Visit safe.com

GET

Modified Response

Display page

GET

Response

jCSRF: A CSRF Defense for Web 2.0 Applications
Overview

User
Visit safe.com
Display page
Submit form

Browser
GET
Modified Response

jCSRF Proxy
GET
Modified Response

Server
GET
Response
Overview

User
Visit safe.com
Display page
Submit form
jCSRF adds token

Browser
GET
Modified Response

jCSRF Proxy
GET
Response

Server
POST
Modified Response

jCSRF: A CSRF Defense for Web 2.0 Applications
Overview

User
Visit safe.com

Browser
Display page
Submit form

jCSRF Proxy
jCSRF adds token
Modified Response

Server

GET
Response

GET
POST

Verify Token
Same-Origin Protocol

User

Visit safe.com

Browser

GET safe.com

jCSRF Proxy

GET safe.com

Response

Bank Server
Same-Origin Protocol

User

Browser

jCSRF Proxy

Bank Server

Visit safe.com

GET safe.com

Set-Cookie: C

Response + jScript

GET safe.com

Response

COOKI

FORM DATA

POST post.php

Cookie: C

Data: form data + P

Submit form

jCSRF

Handler

User

Visit safe.com GET safe.com GET safe.com

Response Set-Cookie: C

Response + jScript

Display page

jCSRF Proxy Bank Server Browser

Register submit 

Handler for forms

POST post.php

Cookie: C

Data: form data + P

Submit form

jCSRF

Handler

User

Visit safe.com GET safe.com GET safe.com

Response Set-Cookie: C

Response + jScript

Display page

jCSRF Proxy Bank Server Browser

Register submit 

Handler for forms

POST post.php

Cookie: C

Data: form data + P

Submit form

jCSRF

Handler

User

Register submit 

Handler for forms

POST post.php

Cookie: C

Data: form data + P

Submit form

jCSRF

Handler

User

Register submit 

Handler for forms
Same-Origin Protocol

User
Visit safe.com

Browser
GET safe.com
Set-Cookie: C
Response + jScript

jCSRF Proxy
GET safe.com
Response

Bank Server
Register submit handler for forms

PO ST post.php
Cooki e: C
Data: form data + P
Submit form
Same-Origin Protocol

User
Visit safe.com

Browser
GET safe.com
Set-Cookie: C
Response + jScript

jCSRF Proxy
Register submit handler for forms

Bank Server
GET safe.com
Response

Display page

PO ST post.php
Cooki e: C
Data: form data + P
Subm it form
jCSRF
Hand ler
User
Visit safe.com
GET safe.com
Response

Set-Cookie: C
Response + jScript
Same-Origin Protocol

User → Browser
- Visit safe.com
- Display page
- Submit form

Browser → jCSRF Proxy
- GET safe.com
- Set-Cookie: C
- Response + jScript

jCSRF Proxy → Bank Server
- GET safe.com
- Response

Register submit handler for forms

User

Browser

jCSRF Proxy

Bank Server
Same-Origin Protocol

User → Visit safe.com
Browser → GET safe.com
 jCSRF Proxy → GET safe.com
 Bank Server → GET safe.com

Register submit handler for forms
Display page
Submit form
jCSRF Handler
Copy C from cookies into form as P

User
Browser
jCSRF Proxy
Bank Server
Same-Origin Protocol
Same-Origin Protocol

User

Visit safe.com

Browser

GET safe.com

Set-Cookie: C

Response

Response + jScript

jCSRF Proxy

POST post.php

Cookie: C

Data: form data + P

C == P

Bank Server

Register submit handler for forms

Display page

Submit form

jCSRF Handler

Copy C from cookies into form as P

Results

CSRF Defenses

jCSRF

Conclusions
Same-Origin Protocol

User

Browser

jCSRF Proxy

Bank Server

Visit safe.com

GET safe.com

Set-Cookie: C

Response + jScript

Display page

Submit form

Register submit handler for forms

jCSRF Handler

Copy C from cookies into form as P

POST post.php

Cookie: C

Data: form data + P

C == P

POST post.php

Data: form data

Response

jCSRF: A CSRF Defense for Web 2.0 Applications
Potential Attack Vectors

- Guess the user’s token.
Potential Attack Vectors

• Guess the user’s token.
• Steal user’s token.
Potential Attack Vectors

- Guess the user’s token.
- Steal user’s token.
- Cause victim to use attacker’s token.
Supporting Cross-Origin Request

- Scenario: Source domain $S$ submits a request to target $T$.
- Problem: $S$ can set authentication parameter $P$ for $T$, but can’t set a cookie for $T$. 
Supporting Cross-Origin Request

- Scenario: Source domain $S$ submits a request to target $T$.
- Problem: $S$ can set authentication parameter $P$ for $T$, but can’t set a cookie for $T$.
- Approach:
  - $T$ gives $S$ a capability
  - $S$ includes this capability into the request for $T$. 

Supporting Cross-Origin Request

- Scenario: Source domain $S$ submits a request to target $T$.
- Problem: $S$ can set authentication parameter $P$ for $T$, but can’t set a cookie for $T$.
- Approach:
  - $T$ gives $S$ a capability
  - $S$ includes this capability into the request for $T$.
- Challenge: capability should be usable only by one specific user.
  - If $S$ requests the capability from $T$ on behalf of the user, $T$ has nothing to bind the user to.
  - Solution: have the browser bind the capability for $T$, then send it to $S$. 

Cross-Origin Protocol

User

Visit safe.com

Browser

GET safe.com

Response + jScript

Display page

Source Proxy (S)

Register submit 

handler for forms

Target Proxy (T)

T iframe

POST post.php

C ook ie: C

Data: form data + P ST

postM essage P ST to S

GET T iframe

Visit safe.com GET safe.com

Response + jScript

Display ... C ook ie: C

P ST

T iframe

Register submit 

handler for forms

P ST

= AES(C || S)

Insert P ST into form

Decrypt P ST

to

check for C and S

User

Submit form

S

jC SRF

handler

GET T iframe

Visit safe.com GET safe.com

Response + jScript

Display ... (T)

T iframe

Register submit 

handler for forms

P ST

= AES(C || S)

I nsert P ST into form

User

Submit form

S

jC SRF

handler

POST post.php

C ook ie: C

Data: form data + P ST

postM essage P ST to S

GET T iframe

Visit safe.com GET safe.com

Response + jScript

Display page

Browser Source Proxy (S) Target Proxy (T)
Cross-Origin Protocol

User
Visit safe.com
Browser
GET safe.com
Source Proxy (S)
Target Proxy (T)
Register submit handler for forms
Display page
Submit form

S
jCSRF handler

GET safe.com
Response + jScript

Insert postMessage to Source

postMessage to Source

Decrypt to check for Source and Target
Cross-Origin Protocol

User
Visit safe.com
Source Proxy (S)
Target Proxy (T)
Browser
GET safe.com
Response + jScript
Display page
Register submit handler for forms
Submit form

jCSRF

POST post.php
C ookie: C
Data: form data + P ST
postM essage P ST to S
GET T iframe
Visit safe.com GET safe.com
Response + jScript
Display page
Browser Source Proxy (S) Target Proxy (T)
Register submit handler for forms

P ST
= AES(C || S)
Insert P ST into form
Decrypt P ST
to check for C and S
Cross-Origin Protocol

User visits safe.com

Browser

Source Proxy (S)

Target Proxy (T)

Register submit handler for forms

Display page

Submit form

GET safe.com

Response + jScript

Register submit handler for forms

GET T iframe

Response + jScript

Display page

POST post.php

Data: form data + PST

Register submit handler for forms

POST to S

GET T iframe

Data: S

Set-Cookie: C

PST = AES(C || S)

Register submit handler for forms

POST to T iframe

X-No-Csrf: Yes

Data: S

Set-Cookie: C

PST = AES(C || S)
Cross-Origin Protocol

User
Visit safe.com
Submit form
Display page
Browser
Source Proxy (S)

Register submit handler for forms
GET safe.com
Proxy (S)

Target Proxy (T)
GET safe.com
Register submit handler for forms

Insert $P^S_T$ into form
POST post.php

Source Proxy (S)
Set-Cookie: C

Target Proxy (T)
Respons + jScript

User
Visit safe.com
Response + jScript
Display page
Browser

Target Proxy (T)
XMLHttpRequest

Register submit handler for forms

Insert $P^S_T$ into form

POST post.php

Source Proxy (S)
Set-Cookie: C

Target Proxy (T)
Respons + jScript
Display page
Browser

Target Proxy (T)
XMLHttpRequest

Register submit handler for forms

Insert $P^S_T$ into form

POST post.php

Source Proxy (S)
Set-Cookie: C

Target Proxy (T)
Respons + jScript
Display page
Browser
Cross-Origin Protocol

User → Visit safe.com → GET safe.com → Response + jScript

Browser

Source Proxy (S)

Register submit handler for forms

Display page

Submit form

Target Proxy (T)

GET T iframe

Visit safe.com GET safe.com

Response + jScript

Register submit handler for forms

POST post.php

Cookie: C

Data: form data + P^ST

Decrypt P^ST to check for C and S

P^ST = AES(C || S)

Set-Cookie: C

Data: S

XMLHttpRequest

X-No-Csrft: Yes

Insert P^ST into form

POST post.php

Cookie: C

Data: form data + P^ST

Decrypt P^ST to check for C and S

P^ST = AES(C || S)

Set-Cookie: C

Data: S
Potential Attack Vectors

• Guess the user’s token.
Potential Attack Vectors

- Guess the user’s token.
- Steal the user’s token.
Potential Attack Vectors

- Guess the user’s token.
- Steal the user’s token.
- Cause victim to use attacker’s token.
Limitations

- jCSRF only protects POST requests.
  - RFC 2616 mandates GET requests free of side effects
  - Usability Issues: cannot type url or bookmark pages
Implementation

- Prototype implemented in Python as a server-side transparent proxy that includes a JavaScript file in every HTML page.
- Adds token to POST forms before submission.
  - Uses jQuery’s *live handlers* to bind the token handler to all forms, including those dynamically created afterwards.
  - Wraps existing event handlers to make jCSRF transparent to them.
Protection

- Correctness is achieved by design, by relying on the Same-Origin Policy to prevent the attack scenarios presented.
- Tested it against two CVE vulnerabilities (RoundCube CVE-2009-4076, Acc PHP eMail CVE-2009-4906).
Compatibility

• Tested with popular Open-Source web applications.
• Future Work: develop a client-side proxy to test it with known web applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Version</th>
<th>LOC</th>
<th>Type</th>
<th>Compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpMyAdmin</td>
<td>3.3.7</td>
<td>196K</td>
<td>MySQL Administration Tool</td>
<td>Yes</td>
</tr>
<tr>
<td>SquirrelMail</td>
<td>1.4.21</td>
<td>35K</td>
<td>WebMail</td>
<td>Yes</td>
</tr>
<tr>
<td>punBB</td>
<td>1.3</td>
<td>25K</td>
<td>Bulletin Board</td>
<td>Yes</td>
</tr>
<tr>
<td>WordPress</td>
<td>3.0.1</td>
<td>87K</td>
<td>Content-Management System</td>
<td>Yes</td>
</tr>
<tr>
<td>Drupal</td>
<td>6.18</td>
<td>20K</td>
<td>Content-Management System</td>
<td>Yes</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>1.15.5</td>
<td>548K</td>
<td>Content-Management System</td>
<td>Yes</td>
</tr>
<tr>
<td>phpBB</td>
<td>3.0.7</td>
<td>150K</td>
<td>Bulletin Board</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Related Work

<table>
<thead>
<tr>
<th></th>
<th>Dynamic Requests Support</th>
<th>Cross-Origin Support</th>
<th>Browser compatible</th>
<th>Cross-Origin Support</th>
<th>No source code required</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoForge</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>X-Protect</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CSRFMagic</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CSRFGuard</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Origin</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>SOMA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>jCSRF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusions and Q&A

• Automatically protect dynamic web applications against POST-based CSRF attacks.
• Low overhead (if implemented as an apache module), compatible with most web applications.
Conclusions and Q&A

• Automatically protect dynamic web applications against POST-based CSRF attacks.
• Low overhead (if implemented as an apache module), compatible with most web applications.
• Questions?