Cross-site request forgery (CSRF)

Emanuele Rocca

Vrije Universiteit Amsterdam
Advanced Topics in Computer and Network Security

December 13, 2011

How complicated is it?

```
<img src='http://www.example.org/logout.php'>
```

How serious is the problem?

- #12 in CWE/SANS *Top 25 Most Dangerous Software Errors* list
- the US Department Of Homeland Security considers CSRFs attacks more dangerous than most buffer overflows
- often overlooked, does not receive the same level of attention of other vulnerabilities
- ask David Airey. A CSRF vulnerability in Gmail allowed an attacker to gain control of davidairey.com

How serious is the problem?

Sponsored by
DHS National Cyber Security Division/US-CERT

NIST
National Institute of Standards and Technology

National Vulnerability Database
automating vulnerability management, security measurement, and compliance checking

Vulnerabilities | Checklists | Product Dictionary | Impact Metrics | Data Feeds | Statistics

Home | ISAP/SCAP | SCAP Validated Tools | SCAP Events | About | Contact | Vendor Comments

Mission and Overview

NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

Resource Status

NVD contains:
31631 CVE Vulnerabilities
161 Checklists
146 US-CERT Alerts
2233 US-CERT Vuln Notes
3259 OVAL Queries
Last updated: 07/03/08
CVE Publication rate:
14 vulnerabilities / day

Email List

NVD provides four mailing lists to the public. For information and subscription

National Cyber-Alert System

Vulnerability Summary CVE-2007-1332

Original release date: 3/7/2007
Last revised: 3/9/2007
Source: US-CERT/NIST

Overview

Multiple cross-site request forgery (CSRF) vulnerabilities in TKS Banking Solutions ePortfolio 1.0 Java allow remote attackers to perform unspecified restricted actions in the context of certain accounts by bypassing the client-side protection scheme.

Impact

CVSS Severity (version 2.0):
CVSS v2 Base score: 9.3 (High) (AV:N/AC:M/Au:N/C:C/I:C/A:C) (legend)
Impact Subscore: 10.0
Exploitability Subscore: 8.6

Access Vector: Network exploitable
Access Complexity: Medium
Authentication: Not required to exploit
Impact Type: Provides administrator access, Allows complete confidentiality, integrity, and availability violation , Allows unauthorized disclosure of information , Allows disruption of service

References to Advisories, Solutions, and Tools

How is it different from Cross-site scripting?

XSS = Cross-site scripting

- XSS is all about code injection
- CSRF is about making the user's browser do nasty things

They exploit two different kinds of trust relationships:

- XSS exploits the user $\rightarrow$ site trust
- CSRF exploits the site $\rightarrow$ user's browser trust

Outline

What?

- *confused deputy attack* against a Web browser
- affects sites that rely on a user's identity
- exploits the site's trust in that identity
- trick the user's browser into sending HTTP requests to a target site
- involve HTTP requests that have side effects

How?

- a malicious website instructs a victims browser to send a particular HTTP request to an honest site
- the request appears like a legitimate action performed by the user
- leverages the victim's network connectivity and the browser's state, such as cookies
- disrupts the integrity of the victim's session with the honest site

In practice

- GET requests via image tags
- POST/PUT/DELETE requests via JavaScript form submissions
- can be combined with XSS to broaden the impact

Why?

- reach machines behind a firewall
- confuse services relying on IP address authentication
- exploit websites that rely only on browser state, such as cookies or HTTP basic access authentication
- set cookies

Login CSRF

- particular example of CSRF vulnerability
- mutates browser state, not server-side state
- the attacker forces the victim to log in on a given website

Why are login CSRFs dangerous?

- steal credit card info on PayPal
- read search history on Google
- often overlooked

PayPal

- Alice visits Eve's site and chooses to pay using PayPal
- Alice redirected to PayPal and logs in
- Login CSRF attack: Eve logs Alice in with another PayPal account under Eve control
- Alice enrolls her credit card, but the details are added to the Eve's account

Google.com/searchhistory

- search queries contain sensitive details about the users interests and activities
- logging the user into the search engine as the attacker
- user's search queries are then stored in the attackers search history
- the attacker can retrieve the queries by logging into her own account

Mitigations: isn't the Same Origin Policy enough?

- the Same Origin Policy limits DOM access to scripts from the same 'origin'
- origin = (domain name, protocol, tcp port)
- evil.com cannot read cookies from homebanking.com
- **making requests** to other origins is allowed!

Existing techniques: server side

Websites should implement one or more of the following approaches to defend against CSRF attacks:

- validate a secret token
- check the HTTP Referer header
- include additional headers with XMLHttpRequest

Existing techniques: client side

- logout from websites when you are done with them, avoid "remember me" features
- browse in *private mode*, aka porn mode. All modern browsers support it
- use browser plugins such as RequestPolicy or CsFire (CeaseFire)

CsFire

- blocking all cross-domain traffic is the most secure policy, but would degrade user experience
- CsFire is a browser plugin for Firefox that inspects the outgoing HTTP requests and decides which action to perform on them:
  - block
  - allow
  - strip authentication information

CsFire: default policy and remarks

- POST: always strip
- GET: accept if it is user initiated and with no parameters, strip otherwise
- does not alter the user experience when it comes to AJAX or Single Sign On
- can be inadequate on mashup sites relying on implicit authentication to construct their content

Conclusions

- CSRF vulnerabilities are a serious threat
- it is not simple to act on the client side, mainly because of mashups

Here is what you should do, as a developer:

- read the HTTP 1.1 RFC
- use a serious web framework like Ruby on Rails or Django rather than re-inventing the wheel, possibly making it square
- think about what would happen if requests to your web application were not really initiated by the user