AppSec Blog

23 Jan **2015**

Demystifying Cross-Site Request Forgery

2 comments Posted by SANS Institute
Filed under CSRF, Spot the Vuln

Continuously ranked in the OWASP Top Ten, a large majority of the development community still doesn't understand Cross-Site Request Forgery (CSRF). After years of penetration tests and code reviews, my experiences show that a high percentage of applications, especially new applications, do not have proper CSRF protections in place. This post provides a refresher on CSRF and provides a common defense for this issue.

The Exploit

CSRF occurs when an application trusts that all requests originating from the user's browser are user-directed actions. Imagine that you are logged into your bank's online portal. The application requires users to authenticate and passes a session cookie back to the browser. Subsequent requests made to the banking site must contain the session cookie, allowing the site to identify the user and perform the requested action.

What if an attacker could send a fake request using the victim's browser?

Suppose an attacker wants to transfer money from a victim's account to their own account. In an unprotected site, the attacker simply makes a valid transfer between accounts in the web site and captures the request.

The following snippet shows a hypothetical request to transfer funds:

GET /transfer.jsp?from=attacker&to=attacker&amount=1 HTTP/1.1
Host: bank.com
Cookie: JSESSIONID=aa968b1d26a891d451e02c631553d2f8f56446bd;

The attacker could then generate a malicious request, such as the src property shown in the image tag below:

**

To execute this attack, our attacker uses a little social engineering and sends our victim two phishing emails. The first tells the victim about some malicious activity on their banking account. The second adversities a free vacation, and contains a link to an evil web page containing the image shown above.

The victim views the emails and quickly signs in to the banking web site. After verifying everything looks ok, the victim clicks the link for a free vacation. As long as the victim is still authenticated to the banking site, the browser will conveniently include the active banking session id in the request:

GET /transfer.jsp?from=victim&to=attacker&amount=1000000 HTTP/1.1
Host: bank.com
Cookie: JSESSIONID=2a63ad7a0ce1ab65ecc293f622587c224e8d4e21;

Notice that this looks identical to the original request, except the session id from the victim's browser has been

RSS

Search

Categories

- .Net (22)
- Architecture (8)
- Ask the Expert (9)
- Authentication (11)
- Authorization (4)
- Certification (1)
- Clickjacking (2)
- CSRF (3)
- Database (2)
- defense (21)
- DoS (2)
- encryption (6)
- GWEB (1)
- honeypot (2)
- iOS (9)
- ipv6 (1)
- java (13)
- mobile (8)
- OAuth (2)
- Passwords (1)
- Pentest (6)
- php (8)
- Secure SDLC (18)
- Security Awareness (4)
- Sessions (4)
- Spot the Vuln (66)
- Top25 (27)
- Validation (1)
- Webcasts (9)
- XSS (2)

Recent Posts

- Breaking CSRF: ASP.NET MVC
- HTTP Verb Tampering in ASP.NET
- ASP.NET MVC: Secure Data Storage
- ASP.NET MVC: Secure Data Transmission
- Breaking CSRF: Spring

automatically included in the request. From the server's point of view, this looks like a valid, user-directed request and the transfer succeeds!

The Defense

CSRF defenses commonly use synchronization tokens. This solution adds a random token to each request that modifies data or performs a transaction. As the request is processed, the application verifies that the value received in the request matches the value generated on the server. The following snippet shows what the transfer request protected with a CSRF token could look like. Notice that we have switched the GET request to a POST request. It is a best practice to make all data modifications and transactions using a POST request, rather than a GET request:

POST /transfer.jsp HTTP/1.1
Host: bank.com
Cookie: JSESSIONID=2a63ad7a0ce1ab65ecc293f622587c224e8d4e21;

from=victim&to=attacker&amount=1000000&token=444a6130-9d2a-11e4-bd06-0800200c9a66

The new token parameter has been added to the POST parameters. As long as the value is sufficiently random, the attacker will be unable to guess the victim's current value as the attack is taking place. If the application properly verifies the token before processing the transfer, the attack will fail!

OWASP's CSRFGuard is a popular Java EE filter that can be configured to protect an entire web site from CSRF attacks by automatically adding and verifying CSRF tokens.

To learn more about protecting your Java and web-based applications from CSRF using OWASP CSRFGuard, sign up for DEV541: Secure Coding in Java!

References:
https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

About the Author
Steve Kosten is a security consultant at Cypress Data Defense and the Denver Chapter President of the Open Web Application Security Project (OWASP) that focuses on information security education related to software applications. He was the co-organizer of AppSec USA 2014. He is an application security specialist who reviews software applications for top 100 firms across multiple industries including the financial, defense, identity management and more. He has a Masters degree in Information Security, and is CISSP and CISM certified.

Permalink | Comments RSS Feed - Post a comment | Trackback URL

2 Comments

Posted January 24, 2015 at 9:53 AM | Permalink | Reply

Sumo

Just keen to understand how 2 FA or Transaction PIN a deterrent to CSRF attack

Posted February 1, 2015 at 4:18 AM | Permalink | Reply

johabell

A well written article. Thank you so much.

Security and Thymeleaf

Archives

Select Month

Links

- Log in
- Entries RSS
- Comments RSS

Post a Comment

***Name**

***Email**

Website

***Comment**

Captcha

***Response**

Post Comment

* Indicates a required field.

Latest Blog Posts

Breaking CSRF: ASP.NET MVC
February 22, 2016 - 1:00 AM

HTTP Verb Tampering in ASP.NET
January 06, 2016 - 4:16 PM

ASP.NET MVC: Secure Data Storage
November 12, 2015 - 4:06 AM

View More »

Latest Tweets @sansappsec

#FBF Security Leadership: Strategies for Success with @fykim [...]
March 11, 2016 - 1:25 PM

#TBT Risky Business: Evaluating the True Risk to your Securi [...]
March 10, 2016 - 9:25 PM

Last day to save $400 at SecWest in San Diego! Register for [...]
March 9, 2016 - 7:24 PM

Latest Papers

Protection from the Inside: Application Security Methodologies Compared
By Jacob Williams

Web Application Firewalls
By Jason Pubal

Protecting Access to Data and Privilege with Oracle Database Vault
By Pete Finnigan

View More »

"Provided a wealth of knowledge and practices to employ when developing Java applications."
- Nicholas Glantz, Lockheed Martin

"Wow, I had no idea, and I've been doing this for close to a decade."
- Michael Knopf, Abacus Technology

"Great overview of different types of attacks and how to mitigate them."
- Tim Sargent, Kinecta Federal Credit Union