WHY CSRF WORKS

To explain the root causes of, and solutions to CSRF attacks, I need to share with you the two broad types of authentication mechanisms used by Web applications:

1. Implicit authentication
2. Explicit authentication

Implicit authentication by Web browsers

Implicit authentication by Web browsers occurs when the browser automatically includes authentication information in HTTP requests; in other words, the Web browser itself is responsible for tracking the authenticated state. Widely used implicit authentication mechanisms in the browser include:

- **HTTP authentication:**This enables the Web server to request authentication credentials from the browser in order to restrict access to certain Web pages. In all the three methods (basic, digest and NTLM), the initial authentication process undergoes the same basic steps. Figure 3 shows a simplified version of the authentication process. If the client requests further restricted resources that lie in the same authentication realm, the browser includes the credentials automatically in the request.

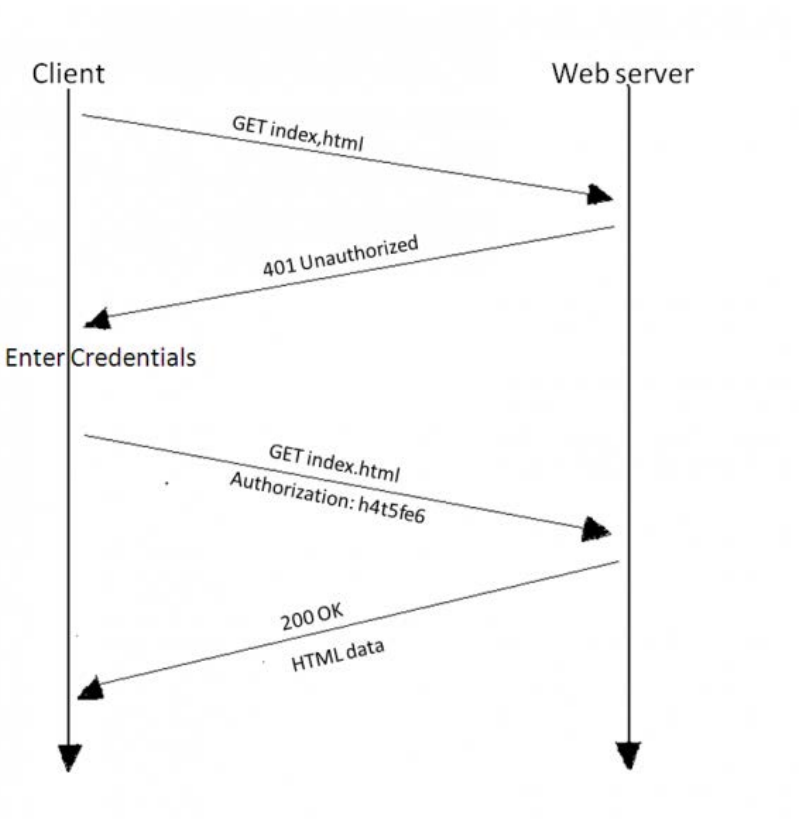

Figure 3: Typical HTTP authentication

- **Cookies:**Web browser cookie technology provides persistent data storage on the client side, which is often used by today's Web applications to store authentication tokens. After a successful login procedure, the server sends a cookie to the client. Every subsequent HTTP request that contains this cookie is automatically regarded as authenticated. The typical cookie authentication process is shown in Figure 4.

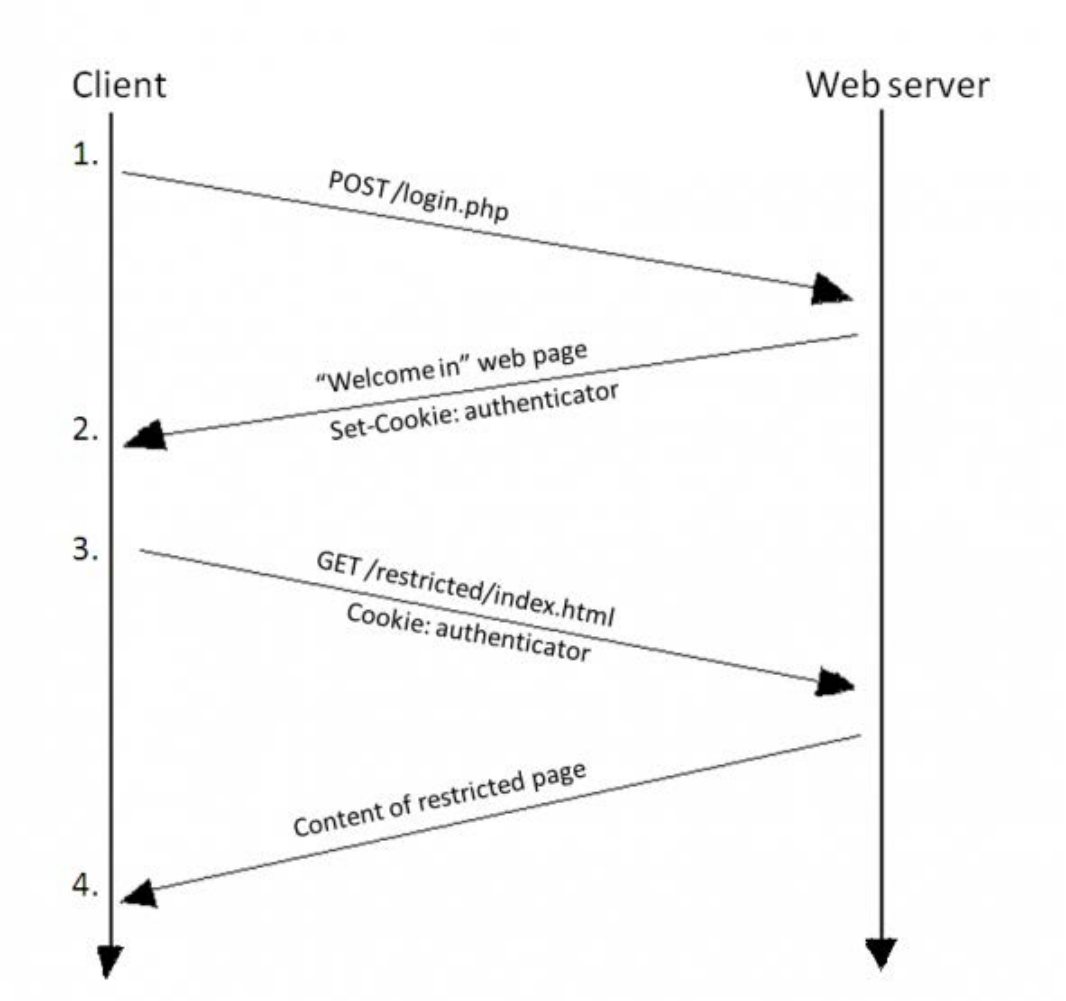

Figure 4: Typical cookie authentication

- **Client-side SSL authentication:** The Secure Socket Layer (SSL), and its successor, the Transport Layer Security (TLS) protocol, enable cryptographically authenticated communication between the Web browser and the Web server. To authenticate, X.509 certificates and digital signature schemes are used.

What all these schemes have in common is that after a successful initial authentication, tokens are sent automatically in further requests, without asking the user for permission — this is what makes Web applications that use these authentication techniques, vulnerable to CSRF attacks.

Implicit authentication by IP address

This special case of implicit authentication is often found on intranets. Here, the authentication is based on certain IP (or MAC) addresses from which requests are made to the application. Shown in Figure 5 is a typical IP-based authentication, in which only users within the intranet are allowed to access the intranet server, and requests from all other IP addresses are denied access.

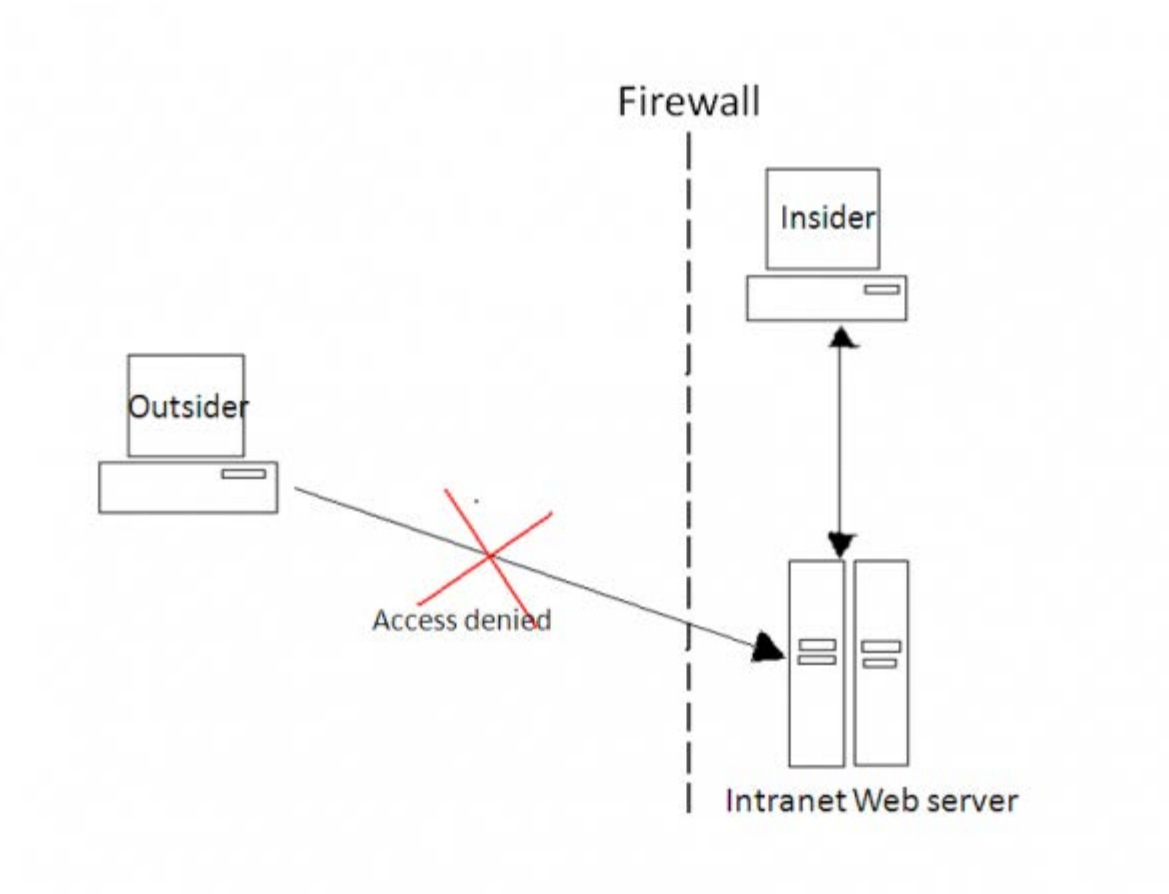

Figure 5: Typical IP-based authentication

Explicit authentication is safer

In this type of authentication, the Web application (or the Web server) is itself responsible for tracking the authenticated state of the user. This is generally done in two ways:

a. URL rewriting, in which the session tokens are included in the URL for every request sent to the server; the URL is generated by the Web application/Web server, and does not require the browser to pass authentication tokens along with the request.
b. Form-based session tokens, in which hyperlinks are replaced with HTML forms that contain session identifiers in hidden form fields.

Though explicit authentication is immune to CSRF, it has other problems, which we will discuss in a while.

The main reason that CSRF works is that Web applications using implicit authentication mechanisms do not verify that a state-changing request was created within the Web application. Because of the implicit authentication of the user, the attacker can easily control the user's session.

Another underlying factor that enables CSRF attacks is the application's use of predictable URL/form actions in a repeatable way.

Some myths about CSRF	
Myths	**Facts**
CSRF is just a special case of XSS	XSRF is a separate vulnerability from XSS, with a different solution. XSS protection won't stop XSRF attacks, though it is also important to

Some myths about CSRF	
Myths	**Facts**
	guard against XSS on a priority basis.
Applications aren’t vulnerable to CSRF if they use multi-page forms to perform actions.	While multi-page forms certainly make exploitation harder, attackers usually can exploit them. Attackers frequently use multiple IFRAMEs when building multi-page form CSRF exploits.
CSRF is solvable by forcing all sensitive requests over POST while denying GETs.	While POSTs can be more difficult to exploit, they certainly are exploitable, as shown in Scenario 1. Using POST rather than GET can provide a defense only against local CSRF attacks; POST requests can still be created with hidden IFRAMEs on foreign Web pages. Forms can mislead users about what they are sending, and where, and scripting can lead to automatic submission. JavaScript is fully capable of sending POST requests via form submissions.
CSRF can be prevented by filtering based on the referrer header.	This option is very unreliable, since attackers can easily block the sending of the referrer header, through the use of certain browser and Flash exploits. Some browsers also omit the referrer header when they are being used over SSL. Moreover, many firewalls and anti-spyware software often drop referrers, in their default

Some myths about CSRF	
Myths	**Facts**
	mode, without letting users know. But, using referrers can be viewed as another incremental roadblock.
Browser enforcement of Same Origin Policy (SOP) prevents CSRF.	Browsers that implement SOP will prevent scripts from accessing the DOM of a page originating from another domain, or accessing cookies that originate from other domains. However, they do not prevent scripts from sending requests to other domains. Furthermore, when a script sends a request to another domain (or when an IMG tag's SRC attribute is set to another domain), the browser will execute the request, and will also send any cookies it has that are valid for the domain, along with the request. (See the following highlight on SOP).

Sane Origin Policy (SOP): Web browsers use a security model called the same origin policy (SOP) to enforce some access restrictions on Web applications. The SOP identifies each website using its origin, which is a unique combination of protocol, domain and port, and creates a context for each origin. Two resources are considered to be of the same origin only if all these values are exactly the same.

Source : http://www.opensourceforu.com/2010/11/securing-apache-part-3-xsrf-csrf/