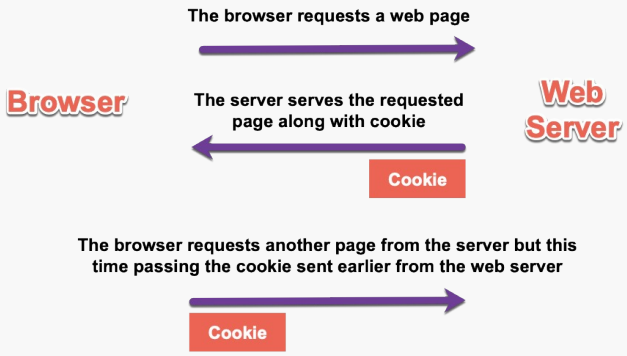
Persistent and Non-persistent cookies in ASP.NET

Published 9/22/2019 by Raghav Khunger

Cookies are the files that reside on your computer. Cookies are created on your machine when you access a website. They are stored on your computer hard disk or in browsers session so that they can be accessed by web servers until they are deleted or have expired. They are used to store pieces of information about your interactions with the website you accessed. The web server can use this information later for the subsequent requests.

Here is a simple workflow to explain that:



Cookies can have an expiration date which can decide how long the cookie will reside in the browser. A cookie may persist only for the lifetime of a user session.

There two types of cookies:

Persistent:

These cookies are stored on your computer hard disk. They stay on your hard disk, accessed by web servers until they are deleted or have expired.

Example:

```
public void SetPersistentCookies(string name, string value)
{
    HttpCookie cookie = new HttpCookie(name);

    cookie.Value = value;

    cookie.Expires = Convert.ToDateTime("12/12/2008");

    Response.Cookies.Add(cookie);
}
```

Non-persistent:

These cookies are saved only while your web browser is running. They can be used by a web server only until you close your browser. They are not saved on your disk

```
public void SetNonPersistentCookies(string name, string value)
{
    HttpCookie cookie = new HttpCookie(name);

    cookie.Value = value;
```

```
    Response.Cookies.Add(cookie);
}
```

From the RFC (emphasis mine):

> The cookie setter can specify a deletion date, in which case the cookie will be removed on that date.
>
> **If the cookie setter does not specify a date, the cookie is removed once the user quits his or her browser.**
>
> As a result, specifying a date is a way for making a cookie survive across sessions. **For this reason, cookies with an expiration date are called persistent.**
>
> As an example application, a shopping site can use persistent cookies to store the items users have placed in their basket. (In reality, the cookie may refer to an entry in a database stored at the shopping site, not on your computer.) This way, if users quit their browser without making a purchase and return later, they still find the same items in the basket so they do not have to look for these items again. If these cookies were not given an expiration date, they would expire when the browser is closed, and the information about the basket content would be lost.

Only the domain that set the cookie can read the cookie in order to prevent the cross-scripting attack. If a cookie is valid and can be read by the domain, it will be passed along with the HTTP request to the domain that it originated from.

If you want a cookie to expire at a specific time, you need to set an expiration date. Expiry set can be done either at the client-side or at the server side. If you want the cookie to expire when the session ends, don't set an expiration date i.e leave expiration date setter.

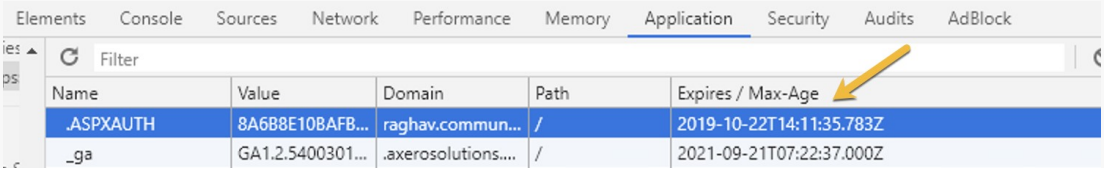
Following is a quick example of checking both of the above cases with a sample code in ASP.NET. Below we will create an AUTH cookie in two different ways:

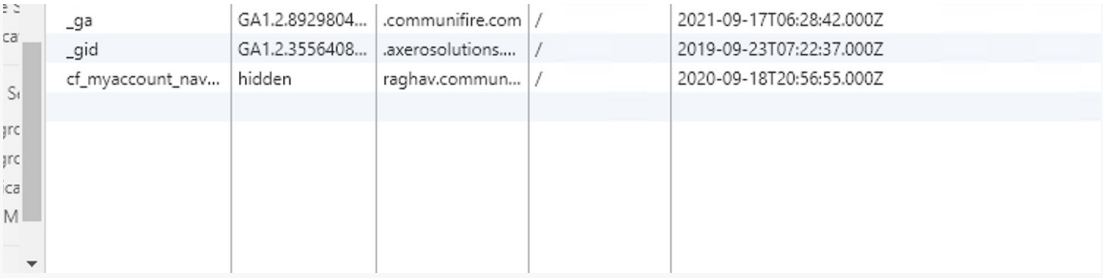
```
public static void AddAuthCookieToResponse(string userName, bool isPersistent, DateTime cookieExpiry)
{

    FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(1, userName, DateTime.Now,
        cookieExpiry, isPersistent, userName);
    string ticketString = FormsAuthentication.Encrypt(ticket);


    HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName, ticketString);

    if (isPersistent)
    {
        // Set the expiry of cookie only when we have to create a persistent cookie.
        cookie.Expires = cookieExpiry;

    }

    if (!string.IsNullOrEmpty(FormsAuthentication.CookieDomain))
    {
        cookie.Domain = FormsAuthentication.CookieDomain;
    }
    cookie.Path = FormsAuthentication.FormsCookiePath;


    HttpContext.Current.Response.Cookies.Add(cookie);

}
```

Setting persistent cookie:

```
AddAuthCookieToResponse(userName: "admin",isPersistent: true, cookieExpiry: DateTime.Now.AddDays(30));
```

Calling the above code will generate a persistent auth cookie which can be confirmed with dev tools:

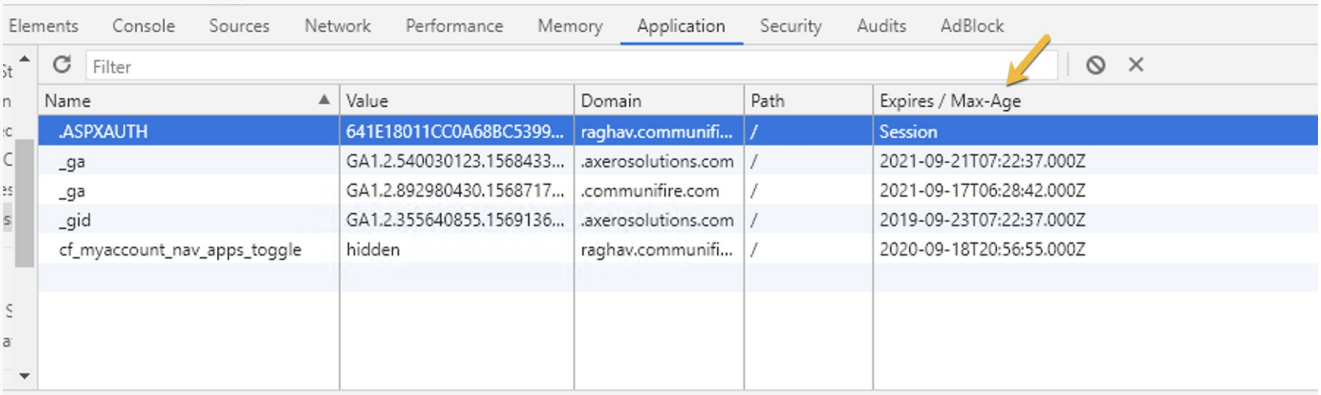




_ga	GA1.2.8929804...	.communifire.com	/	2021-09-17T06:28:42.000Z
_gid	GA1.2.3556408...	.axerosolutions....	/	2019-09-23T07:22:37.000Z
cf_myaccount_nav...	hidden	raghav.commun...	/	2020-09-18T20:56:55.000Z

```
AddAuthCookieToResponse(userName: "admin",isPersistent: false, cookieExpiry: DateTime.Now.AddMinutes(1));
```

The above call will generate a non-persistent cookie (Session cookie) which can be confirmed by checking the expiry of the cookie:



In ASP.NET along with ASPX Auth cookie creation, one more property plays its role and that is **FormsAuthentication.SlidingExpiration** property (The default value of this property is true).

> Sliding expiration resets the expiration time for a valid authentication cookie if a request is made and more than half of the timeout interval has elapsed. If the cookie expires, the user must re-authenticate. Setting the SlidingExpiration property to `false` can improve the security of an application by limiting the time for which an authentication cookie is valid, based on the configured `timeout` value.

In our above example where we added a persistent cookie, we set an expiration date of 30 days. On the first request, a cookie will be generated and will be stored on the client's machine for 30 days. If no request is made in between those 30 days the cookie will be expired. The auth cookie will not be sent to the server with a request which is issued after that 30 days period. The user will see himself as a Guest user and the user will have to re-authenticate to be treated as a logged in user.

With the non-persistent example, you will be wondering why we set an expiration in FormsAuth ticket and not in cookie itself. What is the role of setting the expiration in FormsAuth ticket?

The answer for those questions is that the 1 minute (expiration of Session cookie) which we set there for the FormsAuth ticket is the time after which the Session cookie will be expired and will be not be sent to the server along with subsequent requests. If any further request is made before that 1 minute expiry time the sliding expiration will come into effect and session cookie will be rewritten and the expiry time will be extended for 1 more minute.

Further, from https://stackoverflow.com/a/1637165/148657

1. > Non persistent authentication cookie (no Expires property set). In this case, the cookie will be stored only in the browser's memory and will be lost once the browser is closed. The ticket can have either a fixed timeout or a timeout with a sliding expiration. Once the timeout is reached, despite that the cookie is still sent by the browser, the client will be logged off. For the sliding expiration to work the cookie is rewritten on every request because the ticket changes.

2. > Persistent authentication cookie. In this case, . Same ticket Timeout rules apply here.

tags : asp.net, Cookies