



The browsers also come with a tool called cookie managers to efficiently view and manage the cookies. A Cookie Manager is installed along with the browser and can be used to set preferences about what cookies will be accepted or rejected by the web browser. Figure 1.2 shows the cookie manager which comes with the Mozilla browser.

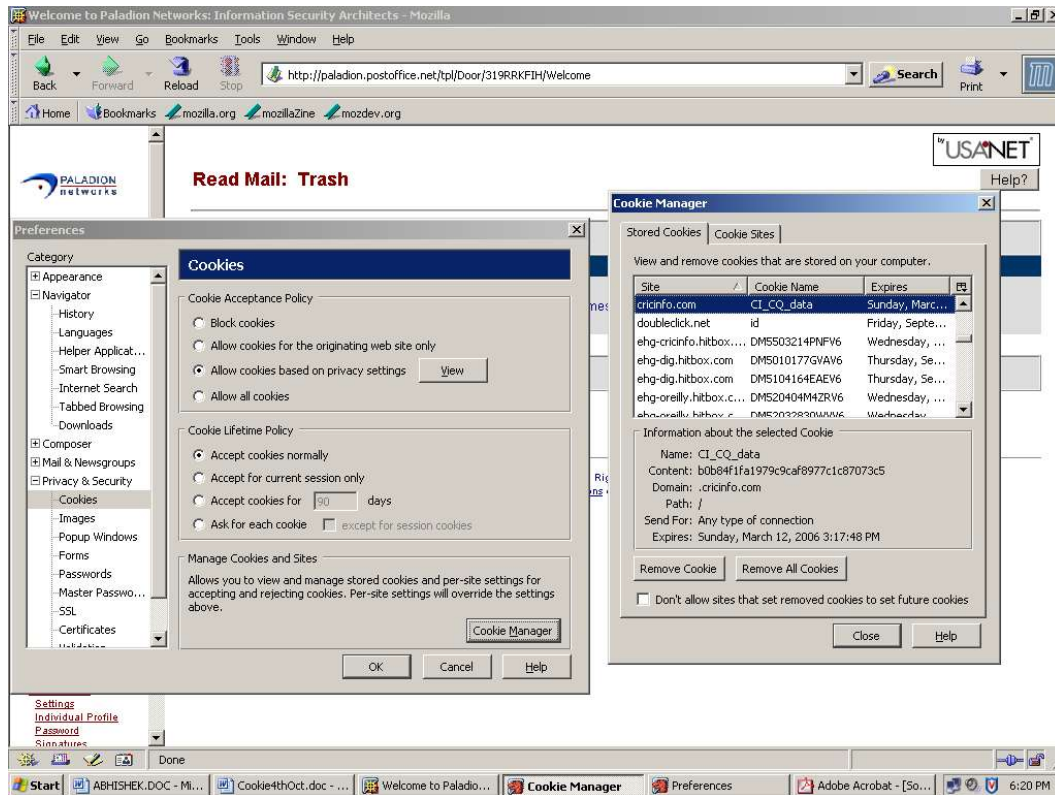


Figure 1.1

## Cookie Composition

A cookie is composed of a number of attributes. To understand the structure of a cookie we now take a look at a popular email service. Here's a snippet of the cookie which is sent by Gmail when a user logs in,

[Set-Cookie: GV=; Domain=mail.google.com; Expires=Sun, 09-Oct-05 13:47:42 GMT; Path=/mail](#)

A typical cookie has primarily 5 parameters,

- Name of the cookie
- Value assigned to the cookie
- Expiration date of the cookie
- Domain the cookie is valid for
- Path the cookie is valid for

In the example above, **GV** is the name of the cookie, which is valid for all URL's under **/mail** at the domain **mail.google.com**. Apart from the parameters mentioned above a cookie have additional parameters such as **secure** and **HttpOnly** which were added to enhance the security and will be explained later.

## Security Risks

Having seen an example of a cookie let us now understand the possible threats associated with the usage of cookies and the mitigation measures that can be implemented.

### Incorrect Domain Settings

We earlier discussed the domain field in the cookie and its purpose. The cookie which we took as example was a valid cookie that worked only for its current domain. However if a web designer accidentally sets the domain field of a cookie to a value of **.com** or **.edu** it would mean that the cookie will become valid for a large chunk of the Web which had websites ending in **.com** or **.edu**. This is the reason all cookies that have **.com** or **.edu** or **.net** are disabled by default as per the RFC 2965 which speaks about HTTP state management. The risk however is still valid incase of cookie mismanagement by websites. We take the following example to understand the risk clearly,

Assume a website named [www.abcfootball.com](http://www.abcfootball.com). When a user logs in to this site a valid cookie for [public.abcfootball.com](http://public.abcfootball.com) is assigned to the user. However one page in the website by mistake sets a cookie with the domain field set to [abcfootball.com](http://abcfootball.com) instead of [public.abcfootball.com](http://public.abcfootball.com). If an attacker realizes this and manages to capture this cookie, he now has a cookie which is valid for [public.abcfootball.com](http://public.abcfootball.com), [private.abcfootball.com](http://private.abcfootball.com) and everything else ending in [abcfootball.com](http://abcfootball.com). Using this cookie the user may get unauthorized access to various sub domains of the website.

The solution for this is straightforward. While setting the parameters of the cookies the developers need to make sure that the domain variable is set properly and the cookie follows the RFC standards.

### Incorrect Path Settings

The PATH field in a cookie controls the directories on the server the cookie is valid for. In the earlier example we saw a cookie GV which had the PATH field set to **/mail**. This implies that if a

user had this cookie it would be valid for all directories under /mail only. There would be other mechanisms to prevent him from viewing other user's mailboxes under /mail.

However if this cookie is incorrectly set to / instead of a specific directory, then the cookie may become valid for all directories. A malicious user may be able to manipulate the URL and view the root directory of the mail server. This could potentially result in a lot of sensitive information being leaked to unauthorized users.

The solution for this problem is also similar to the earlier problem. While setting the parameters of the cookies the developers need to make sure that the path variable is set properly.

### **Login using Persistent Cookies**

At times websites use a persistent cookie to authenticate a user. If we take Gmail as an example and check the "Remember me on this computer" box before logging in as shown in Figure 1.3, then Gmail sets a persistent cookie on the desktop used by the user. Next time Gmail is opened the user will be directly logged in without providing username and password. Here's the response from Gmail which is sent on login,

**Set-Cookie: rememberme=true;Domain=.google.com;Path=/;Expires=Thu, 21-Apr-2016 12:03:07 GMT**

The "rememberme" cookie has a name-value pair which contains text -->

Expires: 04/17/2016 05:52:10 PM

Since the "expires" attribute is set to a future date a persistent cookie is set by Gmail. The next time a user visits Gmail, if the rememberme cookie has not expired it'll take him directly to the Inbox.

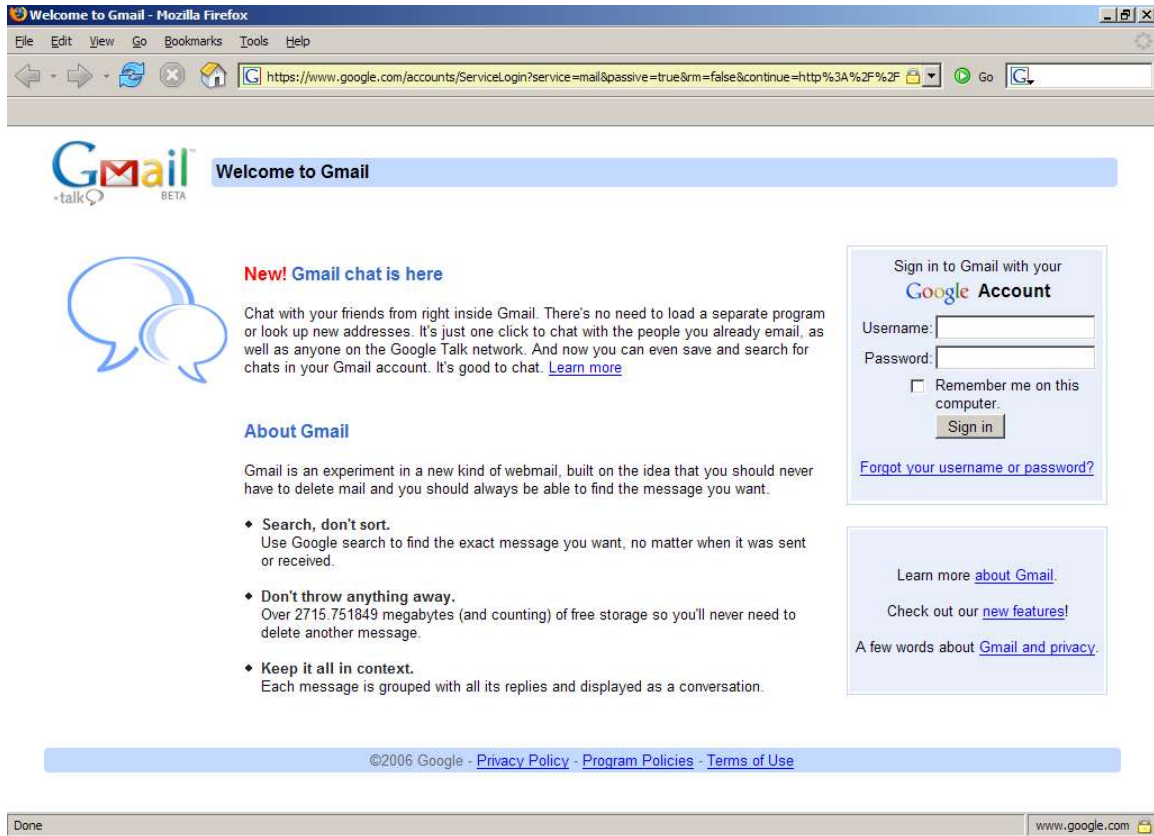


Figure 1.3

Use of persistent cookies for authentication can be a big risk in a shared environment like a cyber café. Assume a scenario where User A checks the “Remember me” box and closes the browser without logging out. If User B uses the same system and has a Gmail account he will be able to clearly see the contents of User A's Inbox. Hence persistent cookies should not be used for authentication of users in a sensitive application. A safer way is to use session ids in non persistent cookies which get destroyed once the user logs out and closes the browser.

### **Sensitive Information in Persistent Cookies**

There's another way in which persistent cookies can give out sensitive data. There could be a scenario where a person buys something from an online shopping website. Now when he submits his payment information online, the site may store his sensitive information such as credit card number offline in a persistent cookie. If the computer is a shared resource another user may be able to access the cookie, obtain the credit card number and use it for malicious purpose.

Storing sensitive information in persistent cookies can be security threat and should be avoided. The users should also take care while accessing sensitive data on a shared computer. Ideally, it is best to avoid accessing sensitive data in a shared environment. Other wise the user should delete

all the cookies by using the browser's cookie manager. Figure 1.4 shows the cookie manager which can be used to clear all cookies stored in browser's cache.

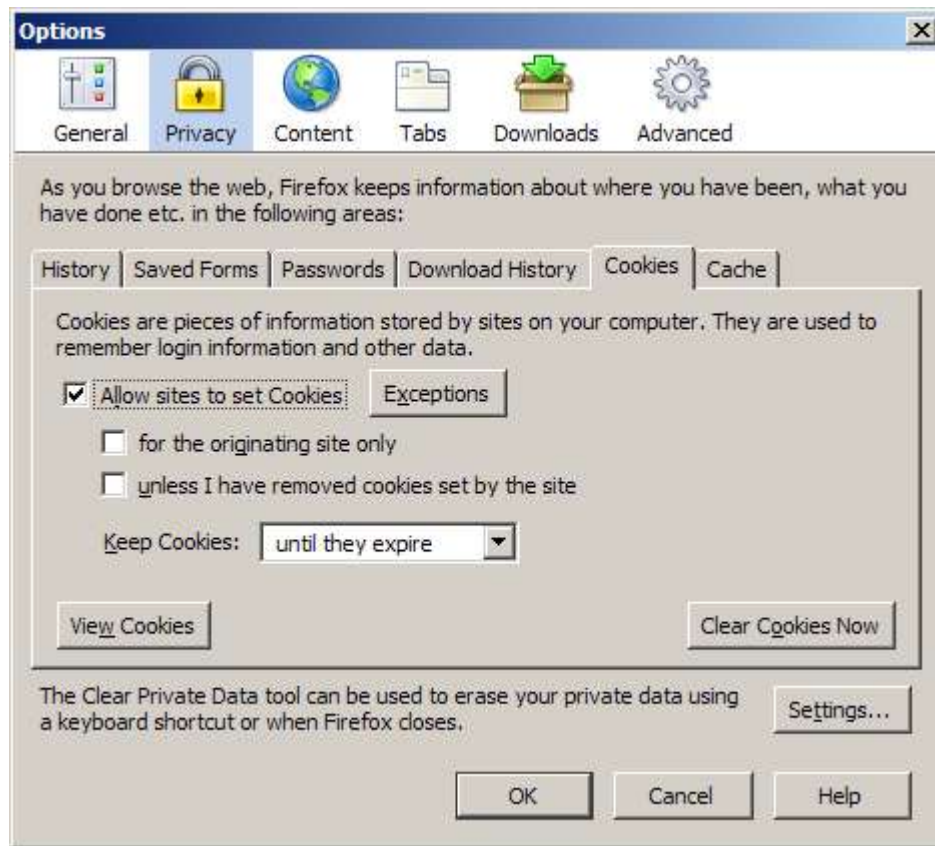


Figure 1.4

### Cookies Transferred in Plain Text

Cookies are often used to store sensitive data such as authentication credentials in the form of session id. If the cookie is sent over a normal http channel then it is transferred in plain text and could get stolen by sniffing on the network. Earlier in the paper we introduced the '**secure**' attribute in cookies. To ensure that the cookie reaches its intended destination without being captured, read or modified along the way, the server should always set the secure attribute in a cookie. Setting of the secure attribute in the cookie ensures that the browser understands that the information being passed is sensitive. It also ensures that the browser make a note of the security level used when the 'Secure' cookie was transferred from the server to the client, which can be

use of SSL. The browser must now maintain at least this standard of security while transferring information back to the server.

### **Caching of Cookies in Caching Proxy Server**

At times a caching proxy server is used which stores the web pages, images etc in its cache, so that the next time the user visits the same page it pulls the page from the cache instead of sending the request to the web server. This saves the user time and reduces load on web server. Whether the page will get cached is controlled by a set of http headers such as cache-control, pragma etc. Though caching proxy servers may improve performance, it may cause a security threat if cookies are also cached. If cookies are cached on a caching proxy server then it could be stolen and used for malicious access.

This vulnerability can be avoided by ensuring that the cookies are not cached on any caching proxy servers while they are sent to the end user. To suppress caching of the cookies following header should be sent in server responses:

`Cache-control: no-cache="set-cookie"`

or

`Cache-control: no-cache="set-cookie2"`

Private pages specific to users also should not be cached and only those which are public and need no authentication can be cached. To suppress caching of a private document at caching proxy server following header should be sent in server responses:

`Cache-control: private`

### **Cross Site Scripting Attack**

Cross Site Scripting is a popular attack which is often used to steal the valid session cookie of a user using a malicious script. These scripts on executing grab the user's cookie from his machine and send it to the attacker who can then use it for malicious purposes.

Various solutions have been suggested for protection against cross site scripting attacks such as input validation. One such solution is to use **HttpOnly** cookies by setting an additional attribute in the Set-Cookie header as follows,

`Set-Cookie: USER=123; expires=Wednesday, 20-Apr-06 23:12:40 GMT; HttpOnly`

Use of HTTPOnly attribute ensures that the cookie cannot be accessed through a script executing on the user's web browser. This means that even if a cross-site scripting bug exists in the application and the browser does not send the cookie to a third party.

## Tracking Cookies

Whenever we visit a website we would expect that only cookies from the originating website are set. This however is not completely true. Many websites contain advertisements from third party companies. On visiting such sites cookies from those companies are also set. Let us look at an example to understand this well. I access my favorite football site [www.abcfootball.com](http://www.abcfootball.com) after clearing out all my cookies. When the page loads there should be cookies relating only to [www.abcfootball.com](http://www.abcfootball.com). But as the page loads up the HTML and Javascript code in the source of the page keeps executing and sets third party cookies. A look at Cookie Manager reveals that after I visit [www.abcfootball.com](http://www.abcfootball.com) I also get a number of other cookies.

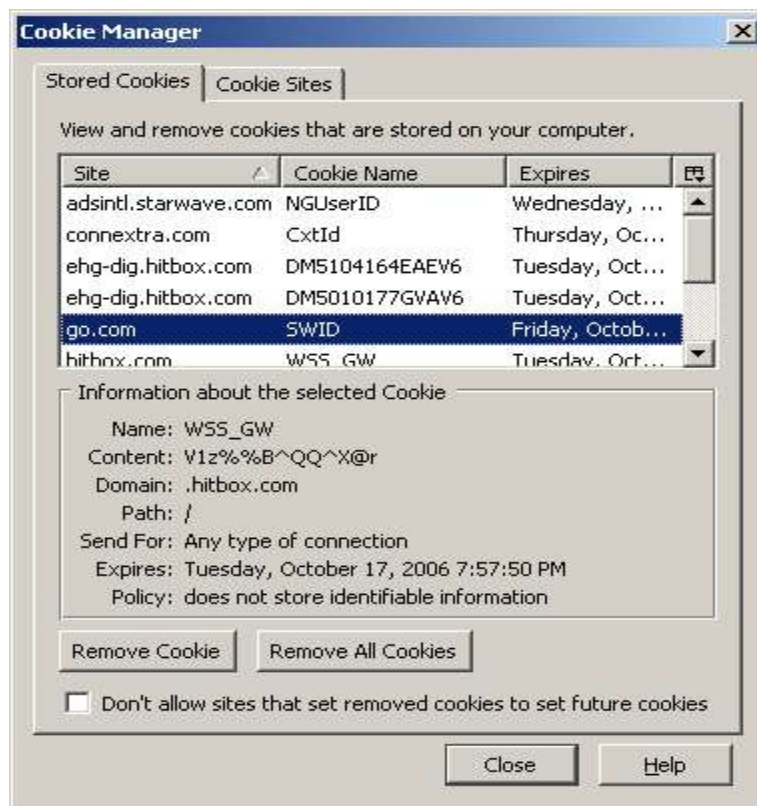


Figure 1.4

A quick look at the HTTP headers when trying to access [www.abcfootball.com](http://www.abcfootball.com) reveals the following,

**Request:**

<http://ehg-dig.hitbox.com/HG?hc=&hb=DM5104164EAE95EN3>

Host: ehg-dig.hitbox.com

Referer: <http://abcfootball.com/origin?camefrom=4765>

**Response:**

HTTP/1.x 302 Moved Temporarily

Server: Hitbox Gateway 8.8.6-rh26 b1

Set-Cookie: CTG=1145605084; path=/; domain=.hitbox.com; expires=Fri, 28-Apr-2006 07:38:04 GMT; max-age=604800

Cache-Control: max-age=0, private, proxy-revalidate

Location: <http://ehg-dig.hitbox.com/HGct?hc=&hb=DM5104164EAE95EN3>

On analysis we see that every visit to [www.abcfootball.com](http://www.abcfootball.com) results in the user internally getting redirected to hitbox.com. This way, Hitbox tracks every user who visits [www.abcfootball.com](http://www.abcfootball.com) and sets cookies on the user's Hard disk which are valid for the Hitbox domain.

There isn't really a risk here because the end user does not lose any sensitive data to a malicious attacker. However it is definitely an invasion of the user's privacy. The reason for this is that when visiting [www.abcfootball.com](http://www.abcfootball.com) the user would expect cookies only of [www.abcfootball.com](http://www.abcfootball.com) to be set on his hard disk. He would not expect his personal browsing habits to be tracked by another party, which is what is happening here.

There isn't a solution for this apart from configuring your browser with great care to stop cookies from all websites being accepted by your browser. The browser can be configured to accept all cookies or block cookies from specific domains.

**Conclusion**

Through the course of this paper we have seen how cookies work and how they can help you make your life easier. However we also saw that accepting cookies does expose you to certain risks and adequate protection must be taken to prevent yourself from falling victim to a hacker's

tricks. Outlined below are a set of good practices for users and developers while playing around with cookies.

### **Developers**

1. The domain and path variables should be set properly for each and every page of the website. Ideally the Path variables should be set to something other than '/'. The Domain variables also should be carefully set to the right domain.
2. Sensitive information such as login information, credit card information etc should not be stored in cookies because if they are captured it can reveal all personal information to the attacker directly.
3. The web server should set appropriate headers to ensure that cookies which store information like session IDs are not cached by a proxy caching server.

### **Users**

1. The users should make sure that they clear the web browser cache and cookies time to time, ideally after each session.
2. Sensitive transactions like Netbanking and credit card related queries should be avoided from public locations or where there are multiple users using the machine.
3. The users should use the cookie manager inbuilt into the web browser to set their preferences and control the websites which are allowed to set cookies.

### **References**

1. [www.faqs.org/rfcs/rfc2965.html](http://www.faqs.org/rfcs/rfc2965.html)
2. <http://www.cookiecentral.com/faq/>
3. <http://computer.howstuffworks.com/cookie6.htm>
4. <http://www.w3.org/Security/Faq/wwwsf2.html#CLT-Q10>
5. <http://www.securityfocus.com/advisories/1083>
6. [http://msdn.microsoft.com/workshop/author/dhtml/httponly\\_cookies.asp](http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp)

Contact Details: arvind.doraiswamy@paladion.net