Stealing passwords via browser refresh

Author: Karmendra Kohli [karmendra.kohli@paladion.net]

Date: August 07, 2004 Version: 1.1

The browser's back and refresh features can be used to steal passwords from insecurely written applications. This paper discusses the problem and the solution. We will show how a bad guy can access the user credentials of the previously logged in user by exploiting this feature, if the web application has not been developed securely

Introduction

Browsers have the ability to maintain a recent record of pages that were visited by a user. The back and forward button on browsers use this functionality to display the pages recently browsed. In addition browsers also keep track of variables that were POSTed to the server while fetching the page.

The refresh feature immensely increases the functionality of the browsers and makes it convenient for users. Moreover it is done transparently so that users do not need to be aware that the variables are automatically posted to the server. All that a user has to do is to click on the "yes" button of a dialog box prompted by the browser before re-posting. This lets a user view the same pages that he had visited before.

Considering functionality, this is a very powerful feature but it can also be used to capture important user credentials from a browser. Here the inherent feature of the browser to store POST variables is exploited to gain access to important user credentials.

We will also be discussing another variation of the attack. These attacks are very simple to execute and require medium level of skills. For each variation of the attack we have proposed the solution used to address the issue.

Capturing the login credentials of a user by refreshing the post login page

Let us consider a user making some online transactions on the Internet.

1. The user types in the URL of the site he wants to visit. In reply the page say “login.asp” is displayed. The web application uses a username and password as the user credentials. This page has form fields “Login ID” and “Password”, like what we have seen in many sites.

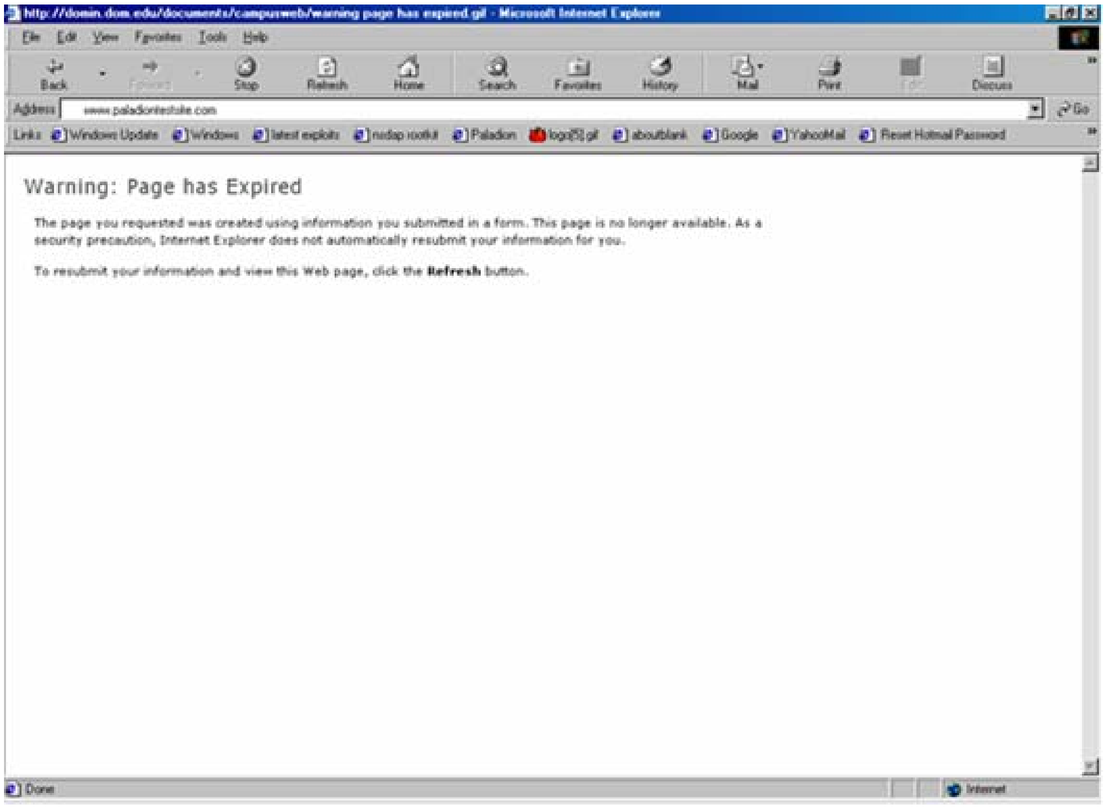
2. The user types his Login ID and password and submits the request to the server. On the server authentication is done by an ASP script, say Myhome.asp, that presents the user with the first page after login, say Myhome.asp.

3. The user browses a number of intermediate pages on the site; say page2.asp, page3.asp, etc.

4. After the user has completed his transactions and finished browsing, he finally clicks on the “Sign Out” button. The logout page, say logout.asp, is invoked which logs off the user. After the logout.asp is displayed on the browser, assume the user leaves the machine without closing the browser window.

5. If a bad guy has access to the same machine as the user, he can see that a logout page is displayed on a browser window.

6. He clicks on the back button drop down list and identifies the page immediately after login – here Myhome.asp. He clicks on the drop down list corresponding to the Myhome.asp page and is displayed the “Warning: Page has Expired” error page that we have seen many a times.


http://domin.dom.edu/documents/campusweb/warning page has expired.gif - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address www.paladiontestsite.com
Go
Links Windows Update Windows latest exploits nsdap rootkit Paladion logo[5].gif aboutblank Google YahooMail Reset Hotmail Password
Done
Internet

Warning: Page has Expired

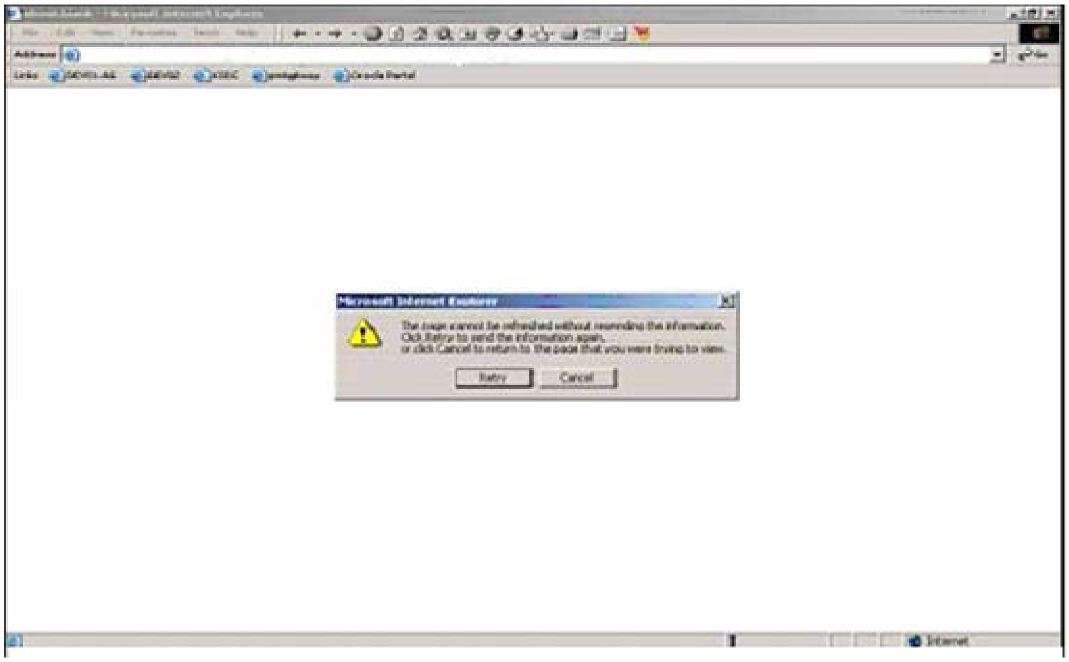
The page you requested was created using information you submitted in a form. This page is no longer available. As a security precaution, Internet Explorer does not automatically resubmit your information for you.

To resubmit your information and view this Web page, click the **Refresh** button.

7. At this point the bad guy starts a browser proxy[1] and configures the browser to go through the proxy.

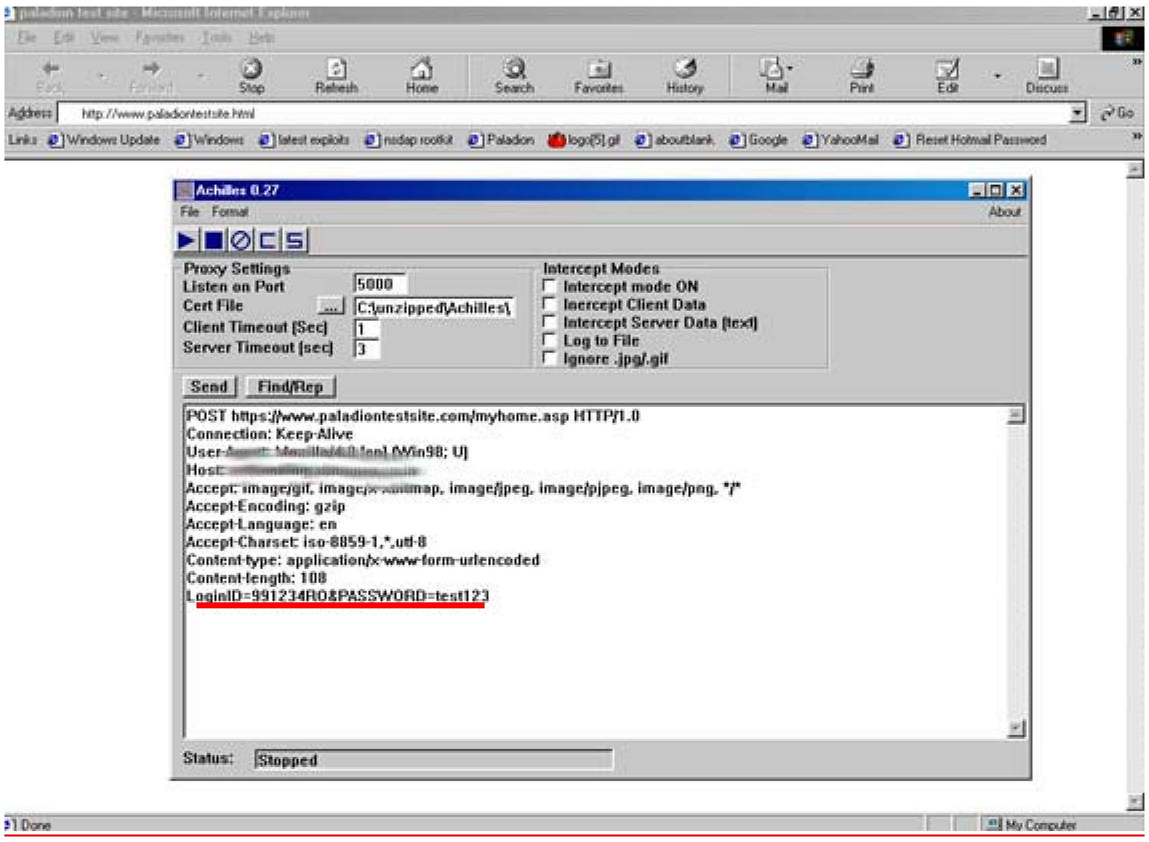
8. On the error page he clicks the refresh button. A pop up warns the user that some of the variables are to be reposted in order for the page to be displayed and asks the user if he wants to continue or not. The bad guy clicks yes.

[1] A browser proxy is a tool that can capture the data flowing between a browser and web server. The browser's proxy settings have to be set to use the proxy. Achilles browser proxy was used during our tests.


Microsoft Internet Explorer

The page cannot be refreshed without resending the information.
Click Retry to send the information again,
or click Cancel to return to the page that you were trying to view.

9. The bad guy views the request sent from the browser to the server in a browser proxy. He is able to see the username and password of the user. He now has complete knowledge of user credentials and hence complete control over the account.


paladion test site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://www.paladiontestsite.html
Links Windows Update Windows latest exploits nsdap rootkit Paladion logo[5].gif aboutblank Google YahooMail Reset Hotmail Password
Achilles 0.27
File Format
About

Proxy Settings
Listen on Port: 5000
Cert File: C:\unzipped\Achilles\
Client Timeout (Sec): 1
Server Timeout (sec): 3

Intercept Modes
- [ ] Intercept mode ON
- [ ] Inercept Client Data
- [ ] Intercept Server Data (text)
- [ ] Log to File
- [ ] Ignore .jpg/.gif

```
POST https://www.paladiontestsite.com/myhome.asp HTTP/1.0
Connection: Keep-Alive
User-[redacted] (Win98; U)
Host: [redacted]
Accept: image/gif, image/[redacted]map, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 108
LoginID=991234RO&PASSWORD=test123
```

Status: Stopped

Cause of the problem

What happens is that each browser instance remembers all pages that it had displayed to the user. In addition to this it also keeps track of all requests that were sent to the server for fetching a particular page. In our case the "Myhome.asp" was displayed only after the user had entered his "Login ID" and "Password". So the browser in addition to remembering that at some moment of time it had served Myhome.asp to the user also remembers the request that was sent for fetching the Myhome.asp – which contains the "Login ID" and "Password" the user had provided on the login.asp page.

Now when the bad guy clicks the refresh button on Myhome.asp, the request that had been used to render the Myhome.asp page is resent to the server. This request contains the "Login ID" and "Password" credentials which the bad guy can intercept.

Solution

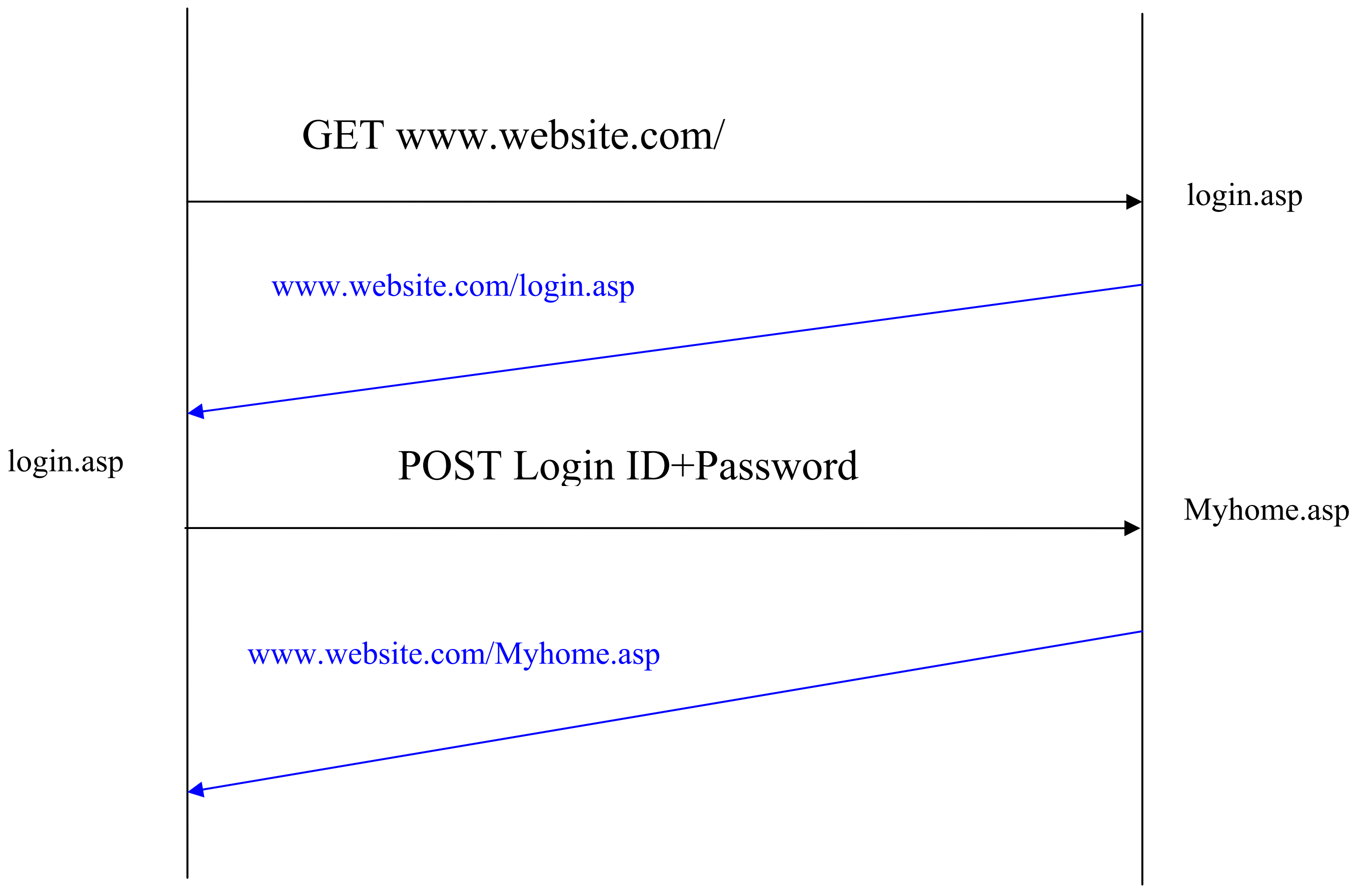
We have listed down two solutions that can be used to solve the above issue. Introduce an intermediate page to carry out the authentication or use the salted hash technique to post the passwords to the server.

1. The recommended solution is to introduce an intermediate page, say Authentication.asp, which performs the authentication checks.

   To understand how that solves the problem, let's first look more closely at what had happened during login.

   - User types URL www.site.com
   - User gets login.asp
   - User submits Login credentials which are send to Myhome.asp
   - User is authenticated by Myhome.asp and is displayed Myhome.asp

The following figure denotes the above steps:



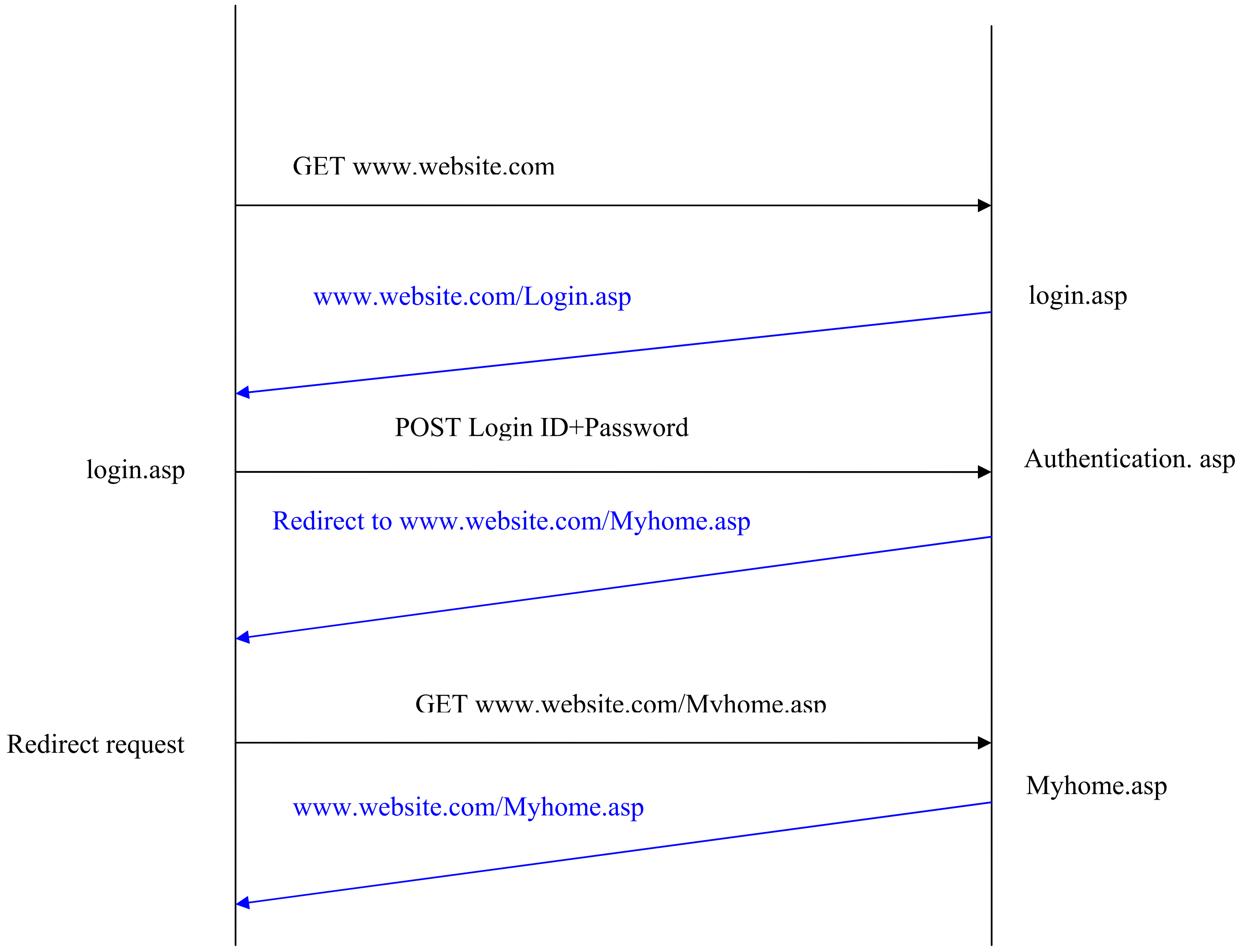
Now, consider that an intermediate page has been introduced. The steps performed are:

- User types URL www.site.com
- User gets login.asp
- User types Login credentials and submits which are sent to Authentication.asp
- User is authenticated by Authentication.asp.
- The Authentication.asp then redirects the user to Myhome.asp.

In this case the user submits the Login ID and password to Authentication.asp. Here the Authentication.asp was never displayed on the browser, hence there is no possibility of the bad guy using the back button to reach the Authentication.asp page. If the bad guy does a refresh on Myhome.asp page, the request which the browser used to render

Myhome.asp is sent. This request is nothing but a redirect request sent by Authentication.asp. Here an important note is that the redirection request must contain the session ID assigned to the user after authentication.

The following figure explains the solution described above:


GET www.website.com
www.website.com/Login.asp
login.asp
POST Login ID+Password
login.asp
Authentication. asp
Redirect to www.website.com/Myhome.asp
GET www.website.com/Myhome.asp
Redirect request
Myhome.asp
www.website.com/Myhome.asp

2. A second solution is to use a salted hash technique for authentication. A discussion on salted hash is outside the scope of this paper; the AppSec FAQ at OWASP has an excellent write-up on salted hash technique.[2]

[2] http://www.owasp.org/documentation/appsecfaq

Variations of the attack

Capturing user credentials by refreshing an intermediate page

The above attack also works for any intermediate page on the application where important user credentials are POST-ed to the server. An example is the password change page. While browsing the site if the user had visited the change password page then the variables posted by him – typically the old password and the newly chosen password - are stored by the browser. If a refresh is done on this page then the request containing these passwords is sent to the server and can be viewed using a browser proxy tool.

Solution

1. The solution is same as discussed earlier. Here too we have to introduce an intermediate page. The password change request should be accepted and processed by this intermediate page. The result of the password change is then displayed to the user by redirection to the next page.

Capturing login credentials when the application uses frames

We discuss a variation of the attack when a site uses frames. Let us consider that in the above example, Myhome.asp consists of two frames. The frame on the top consists of links to other applications and the bottom frame is used to display the results of a request. The frame on top, after getting rendered from the server the first time, remains constant and is not changed while browsing from one page to another. The bottom frame changes for each request.

As the top frame remains constant, a refresh on **ANY** page will resend the request containing the user credentials. Here what happens during a refresh is that both the frames are individually refreshed. The frame on top refreshes and sends the request that was used to render itself. This was the first request from login.asp containing the username and password. Hence it contains the user credentials. The bottom frame refreshes and sends the request generated by its previous page. So we can view the username and password by using a browser proxy.

Solution

1. The solution for this, as discussed above, is to display the Myhome.asp by redirecting through an intermediate Authentication.asp.
2. Another solution as described above is to use the salted hash technique for authentication.

Conclusion

We have seen how simple it is to steal user credentials by using the browser's back and refresh features if a web application has not been developed securely. By implementing the solutions mentioned, we can safeguard the user's credentials from unauthorized access.

References:

Application Security FAQ at Open Web Application Security project
URL: http://www.owasp.org/documentation/faq.html

SecurityFocus HOME Mailing List: Web Application Security
URL: http://www.securityfocus.com/archive/107