[Microsoft Passport Account Hijack Attack]

last major update: 08/10/2001
last minor updates: 23/07/2002
Hacking hotmail and more

Obscure [obscure@eyeonsecurity.net]

EyeonSecurity
http://eyeonsecurity.net/

[Web Applications]

Web Applications are made to extend the usability of the HTTP protocol, along with the efficiency of HTML, JavaScript and so on. The big advantage of Web Applications is that they are immediately accessible, relatively easy to use and can be centrally customized by the developer, at the expense of security issues.

Security issues in Web Applications rise because the HTTP protocol was not designed for Web Applications, and therefore many security measures and extended authentication methods have to be implemented into the Web Application.

[Authentication]

In Web Applications, users have to be authenticated, and will usually be assigned a profile, limited hard disk space, the ability to send data and communicate with other users on the same Web Application, and maybe other Applications and protocols. Many Web Mail Applications, such as Hotmail, make use of cookies to store authentication. For each session at Hotmail, once a user is authenticated, he is assigned a ***random*** cookie, which identifies him from other authenticated users. This way the password does not have to be sent each time he accesses a different page while authenticated. This also means that once the session has ended, the session authentication should expire and therefore a new cookie has to be assigned when the user authenticates himself with Hotmail and starts a new session. All this is described well on the MS Passport site[1].

From now on I will be focusing on Hotmail only, rather than just about any Web Application. However one must understand that the same attacks described here, can be easily adapted for other Web Applications and Web Mail packages, which make use of html, JavaScript and Cookie technology.

[Microsoft Passport]

Microsoft Passport: "A single name, password and wallet for the web". This means that using the same credentials you can access your e-mail (Hotmail), instant messenger service (MSN messenger), calendar (task manager, reminders and so on), and loads of other useful services. All these services are centralized and authenticate with a central system called MS passport. Of course as much as any authorized user can browse without supplying a password from a service which makes use of the Passport technology to another, a cracker (malicious hacker) can do the same without much problems once he gets to look like the authorized user. This kind of service is intended for personal use, so people certainly wouldn't like others to read their e-mail, or view their daily schedule.

[1] Microsoft .NET Passport - http://www.microsoft.com/netservices/passport/

Implementation

Microsoft is trying to build everything around their Network, using Passport authentication. This is complaint with the .NET framework, which allows everything to be seamlessly integrated so that users can jump from one service to another without any problems.

As currently implemented, users can authenticate to Passport via a number of ways:

- Hotmail and Passport sites
- MSN messenger
- MSN Explorer
- Outlook Express
- Other MS applications.

Outlook Express and MSN Explorer make use of Integrated authentication. Hotmail and Passport sites use SSL (HTTPS) to authenticate, and MSN messenger makes use of "MD5" security package.

Once a malicious user gets hold of the session cookie, the above-mentioned authentication methods are useless for services, which rely on the HTTP protocol (such as Hotmail).

Flaws in the design

Previously many exploits inhibited the various Web Browsers, which enabled users to steal cookies from other websites. However this aspect of security in the Passport authentication scheme is supposed to be taken care of by the client user.

To steal the session cookie, the malicious user must either:

- Take hold of the target machine
- Fool the user into sending the session cookie
- Fool the system into sending the session cookie

In this paper I will discuss the 3rd option.

[Fooling the system]

JavaScript allows users to set and retrieve cookies. This is very useful for normal HTTP sites as well as Web Applications. However Web Applications need a lot more control over normal websites. This control is normally achieved through filtering of possibly malicious code in the HTML.

Users do not need permission to send e-mails to authenticated users, giving them the possibility to post data to an authenticated user's mailbox. This is obvious to some extent, since we are talking about e-mail. No one needs authentication to send an e-mail

to a Hotmail account. Therefore the e-mail sent to the Hotmail user has to be treated as non-trusted content.

Hotmail takes very good care to filter out JavaScript, ActiveX and Java applets. Lately it also started checking for images which link to outside the Hotmail account. Having images linking to non-trusted sites means that those sites can easily track the status of the e-mail (if it was read or not). So that a tag in an html mail such as:

```
<img src="http://spam.me.com/tracking.gif">
```

would get filtered by the Hotmail Filtering System. To get around this filtering, one has to just encode the http:// part like &x68;ttp://. 68 is the hex value h, and therefore the Web Browser converts back the encoded value to its original signifier. Of course, the Hotmail filtering system is not working exactly like the Web Browser, and this is where the flaw stands out. However this is not the major issue I am writing about in this document.

Cross Site Scripting

When a logged in user follows an non-trusted link, the Hotmail credentials do not get sent to the non-trusted website. The Hotmail filtering system also takes care to hide the URL of the user's Hotmail account to ensure the user's privacy and maybe to prevent other attacks.

On the other hand, when a user follows an MSN site from a non-trusted e-mail message received through Hotmail web interface, the credentials get sent to the Passport site, and no precautions are taken. This way users following links from a Passport site to another remain authenticated therefore the different services provided by MS Passport operate seamlessly as described earlier.

This also means that if an ASP script, which resides on any MS Passport enabled site allows the user to customize the page (even if not intended) problems will arise.

In my exploit, a user only needs to click on a trusted link and he (or she) will be sending his (or her) credentials to a remote server.

How is this achieved?

To further explain the issue, I will provide an example of a flawed ASP script on an MS Passport site: ErrorMsg.asp, which resides on http://auctions.msn.com/Scripts/

This ASP script can be passed 2 (or possibly more) arguments:

- Source
- ErrMsg

Here we are concerned with ErrMsg argument. This argument allows different scripts to generate different errors and display them to the user in some nice html.

ErrMsg will usually be filled in with something like “User is not authenticated”. Now what if it is filled with **This should be bold.** To my astonishment (at the time of writing this is not fixed yet), I got the HTML tag to work, with no filtering from the ASP script.

To further illustrate this, the url which is passed is actually:

```
http://auctions.msn.com/Scripts/ErrorMsg.asp?Source=0&ErrMsg=<b>This%20should%20be%20bold</b>.
```

If no filtering is done for JavaScript, we can very easily inject our own JavaScript code to retrieve the session cookie stored in the Hotmail user’s browser. Sadly, lately (during the writing of this document), Microsoft seemed to try to fix this by filtering JavaScript (and embedded scripts) tags and entities. This means when the ASP script is passed the following:

- <Script
- Alert
- JavaScript:
- And other commonly used javascript methods

The ASP script simply ignores the input, successfully filtering common Cross Site Scripting attacks.

However Microsoft did not fully patch the issue, so that if HTML encoding were used, the filtering system would not detect the embedded script code, and the code would still be executed.

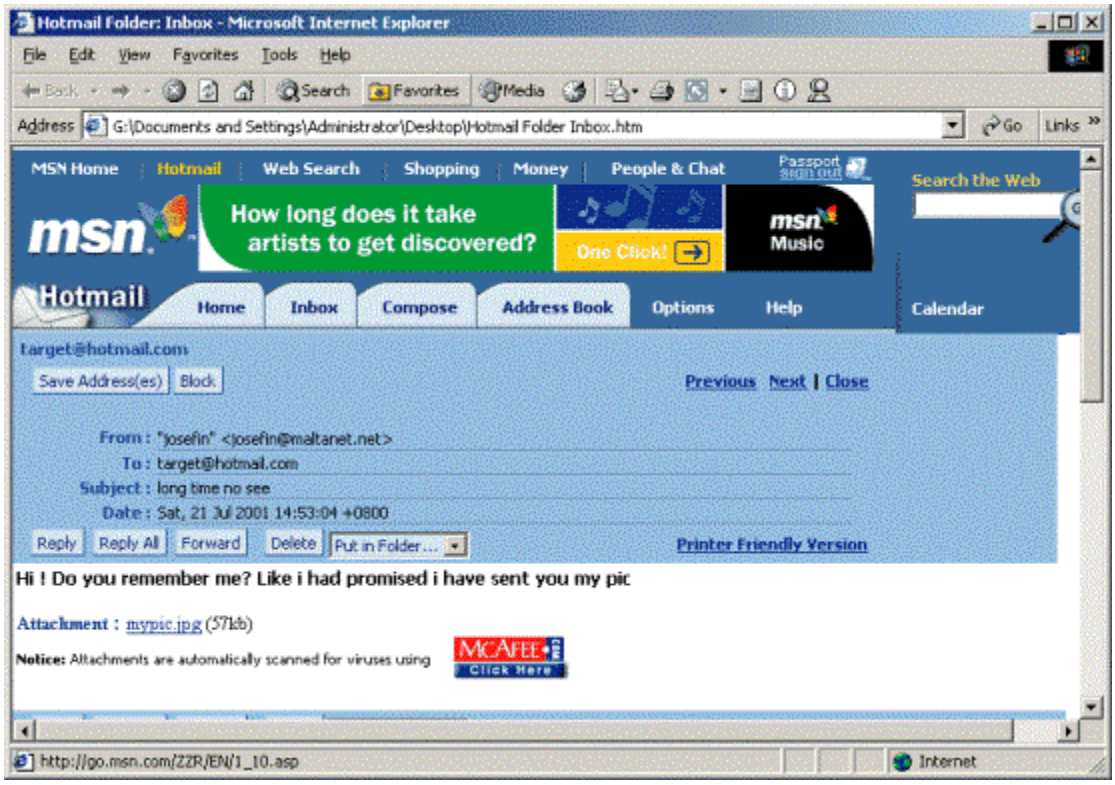
Without any filtering we would just pass the following url and expect a message box to apprear with the MSN cookies:

```
http://auctions.msn.com/Scripts/ErrorMsg.asp?Source=0&ErrMsg=<IMG%20SRC='javascript:alert(document.cookie)'>
```

However to get this to work with the actual ErrorMsg.asp script – we have to encode the URL such as the following:

```
http://auctions.msn.com/Scripts/ErrorMsg.asp?Source=0&ErrMsg=<IMG%20SRC='%26%23x6a;avasc%26%23000010ript:a%26%23x6c;ert(document.%26%23x63;ookie)'>
```

To complete the exploit the malicious user has to send a URL, which actually passes the Cookie to a 3rd party CGI script (probably made by the cracker exploiting this issue) instead of displaying them to the Hotmail user in a Message box. The exploiting e-mail page could look very similar to the one below.



Once the target Hotmail user clicks on the “mypic.jpg” link, he would have sent his credentials to the attacker without asking, any alert or sign that this has actually happened.

[Conclusions]

This paper is based upon trial and error, which means that I do not have access to any source code, and therefore cannot know the actual underlying code that contains the flaw. By the time you read this, Hotmail and MS Passport sites, should have hopefully fixed the described issues.