------------------------------------------------------------------

# -------- Cookie Dethroning.::DEMYSTIFIED::.----------

Unveiling The Facts Of Cookie Manipulation And Hacking Forensic Analysis

--------------- **By.::::.Zer()Kn()ck::.** -----------------------

-------------- **No Patch For Ignorance** ----------------------

******************************************************************

**Part(B)**

## [0x02.1] Unsafe About Cookies:

There are two main areas regarding security around a browser:

--> Reading your private files.
--> Manipulating you into a compromising situation.

A few files provide a lot of information about yourself. These include cache files, the history file, and your bookmarks. By examining someone's cache, history, and bookmarks you can learn a lot about a person. Usually if you are a typical home user running Windows, this is not a problem. But if you are storing your Netscape directory on a server, the server could be compromised and then anything in cache and history is in the hands of someone else. Every access. Submitted forms, including those to change passwords on servers whose service you are paying for.

Being manipulated the other hot area. You can be tricked into supplying user IDs and passwords, reveiling personal information like SS# and credit card information, or even be presented with misinformation to cause you to act in a way to cause a vulnerability to arise. If your browser supports HTML 3.0 extensions and Java, your history file, cache, and other files be plucked from your hard drive. Your machine could be used as a mechanism to attack other resources behind your firewall, sending critical info to an offsite hacker.

## [0x02.2] Vulnerable About History:

The history file-

The default color for a clickable link is blue. Once you've clicked on it and visited the link, it's purple. While the colors may be different depending on what is specified in the HTML, the way your browser keeps track of this information is via the history file.

Since the default is 30 days to expire a link, typically you can see the last 30 days worth of web surfing by examining the history file. "Hmm, Fred keeps looking at a particular set of stocks, does he know something I don't? Hey, Martha keeps looking at lesbian sites, what would her homophobic boss say about that?" Get the idea?

Here's an example -

```
Þ1http://altavista.digital.com/cgi-bin/query?pg=q&what=web&fmt=.&q=
%2B+%2Bmicrosoft+%2Bstock+%2Bprice+%2B+takeover+%2Brumor+%2Bapple ,
_Þ1http://altavista.digital.com/cgi-bin/query?pg=q&what=web&stq=10&
fmt=.&q=%2bapple+%2bmacintosh+%2bhack {_Þ1http://altavista.digital.
com/cgi-bin/query?pg=q&what=web&fmt=.&q=%2Baudit+%2Btrail+%2Bhide
```

If this was from the history file of someone at Microsoft, this might be quite interesting, even valuable.

## [0x02.3]Vulnerable About Bookmarks:

Bookmarks are a problem for the same reason the history file is a problem. It shows what sites you are regularly looking at. If you are bookmarking sites which require passwords to enter, a quick look in the cache will possibly reveil that password, or at least the account ID.

The info gained from here can also be used for social engineering purposes, and can be quite useful. For example, you could determine the user was interested in aquariums and rare fish. This information could be used to assist in guessing a password.

The cache is your browser's way of making things a little easier on your access time, the server your accessing, and the network in general. What happens is that when you access a web page, a copy of the page and any graphics used on that page are stored locally. That way when you access the page again, your browser can pull up the local copy instead of always accessing the network. This saves time and bandwidth. When you reload, your browser compares the cached local file to the one on the server you are accessing and pulls down the latest one. Most browsers will also cache queries and form submittals as well.

If you are looking for dirt on someone, looking for credit card information, or just want to find out what someone's been up to, check their cache. Every query to a search site like AltaVista is stored in cache. Typically every form submittal including accesses to pages requiring an ID and password will be there, unless a site has tagged an HTML document NOT to be cached.

## [0x02.4]Vulnerable About Cache Files:

The cache is typically located in a subdirectory underneath the browser's working directory , usually with the word "cache" in the directory name, depending on your OS and browser version. Otherwise, it may be stored in a temporary directory. For example, IBM's Web Explorer for OS/2 will store its cached files in C:\TCPIP\TMP, and is flushed before each run of the program.

Here is a formatted example from a cache's index file on a Unix workstation, with names changed to protect the not-so-innocent ;-)

```
n b <http://altavista.digital.com/cgi-bin/query?pg=q=web=>
10=.=%2bhack+%2bnt+%2bserver E1
```

```
00/cache31DF458002EC693.cgi
text/html
4 ( <http://www.example.com/user/register.cgi> (r)
rE1 10/cache31DF457002CC693.html

text/html
. " <http://www.example.com/use>
r/welcome.html *1 J 14/cache31DF18940
27C693.html
text/html J
```

Here are three entries. In the first, the user is trying to get NT hacking information from AltaVista. In the second, the user is trying to get signed onto a site called www.example.com, and finally it looks like the user got in. The three cache files are:

```
31DF458002EC693.cgi
31DF457002CC693.html
31DF1894027C693.html
```

You could view these files with a browser, since they're just local copies of the web pages. If 31DF457002CC693.html had a password in it and it was unreadable, you could still do the following:

Access the site yourself and try to login. Check your own cache and replace your cached file with the file 31DF457002CC693.html, renaming it to what your cache file was, and then resubmit the form. If the site is doing only password security, you might get in. If you still don't get in, try substituting the cookie file, as well (in the next section).

The information gained from these sources can also be quite useful for social engineering purposes. For example, you could determine the user was interested in aquariums and rare fish, and use that to assist in guessing a password.

## [0x02.5] Default Browser Holes:

Mosaic 2.2, and all previous version of the NCSA Mosaic for the X Window System had a serious security hole that would allow telnet URLs to execute an arbitrary UNIX command. In Mosaic 2.3 this bug was fixed.

There is a way (involving reconfiguration of both client and server) to have Mosaic execute any arbitrary shell script that is passed over the network. This is a documented feature that cannot be activated accidentally. It must be configured.

An entry is placed in the user or system mailcap that looks like this:

```
        application/x-csh; csh -f %s
```

The client then accesses a document on a local or remote HTTP server that is typed application/x-csh.

Obviously, csh -f will be used as the "viewer" for the document, which means the shell script will be executed on the client's host.

Since Mosaic is not shipped with support for application/x-csh or anything similar in the default settings, this is not a security hole unless you specifically modify your config files to make it so. The same goes for Netscape. You can alter the netscape.ini file and match up certain file types to certain applications, including csh.

I will not go into details regarding what entries should be in the shell script, although mailing the passwd file does seem to be the obvious.

## [0x02.6] Cookie Functions:

It Consists of:
[0x02.6.1] **Function GetCookieVal.**

```
                                    function getCookieVal (offset)
                  {
                                    var endstr =
document.cookie.indexOf (";", offset);
                                    if (endstr == -1)
                                    endstr = document.cookie.length;
                                    return
unescape(document.cookie.substring(offset, endstr));
                  }
```

[0x02.6.2] **Function GetCookieDate.**

```
                                    function FixCookieDate (date)
                  {
                                    var base = new Date(0);
                                    var skew = base.getTime(); // dawn
of (Unix) time - should be 0
                                    if (skew > 0)  // Except on the Mac
- ahead of its time
                                    date.setTime (date.getTime() -
skew);
                  }
```

[0x02.6.3] **Function GetCookie.**

```
                                    function GetCookie (name)
                  {
                                    var arg = name + "=";
                                    var alen = arg.length;
                                    var clen = document.cookie.length;
                                    var i = 0;
                                    while (i < clen) {
                                    var j = i + alen;
                                    if (document.cookie.substring(i, j)
== arg)
                                    return getCookieVal (j);
                                    i = document.cookie.indexOf(" ", i)
+ 1;
                                    if (i == 0) break;
                                    }
                                    return null;
                  }
```

[0x02.6.4] **Function SetCookie.**

```
                                    function SetCookie
(name,value,expires,path,domain,secure)
                  {
                                    document.cookie = name + "=" +
escape (value) +
                                    ((expires) ? "; expires=" +
expires.toGMTString() : "") +
                                    ((path) ? "; path=" + path : "") +
                                    ((domain) ? "; domain=" + domain :
"") +
                                    ((secure) ? "; secure" : "");
                  }
```

[0x02.6.5] **Function Delete Cookie.**

```
                                    function DeleteCookie
(name,path,domain)
                  {
                                    if (GetCookie(name)) {
                                    document.cookie = name + "=" +
                                    ((path) ? "; path=" + path : "") +
                                    ((domain) ? "; domain=" + domain :
"") +
                                    "; expires=Thu, 01-Jan-70 00:00:01
GMT";
                                    }
                  }
```

## [0x02.7] Test To Check For Cookie Support.

**SCRIPT:-**

```
#!/usr/bin/perl
# A Simple cookie test to indicate if the clients browser supports
cookies.
# Distributed through Cookie Central. http://www.cookiecentral.com.

$myLoc = 'http://www.your-server.com/cgi-bin/ct.cgi?TEST';

if($ENV{'QUERY_STRING'} eq 'TEST')
{
   if($ENV{'HTTP_COOKIE'} =~ /Cookie=Test/)
   {
      $reply = "<HTML><TITLE>Good Cookie!</TITLE><BODY>Your browser
supports
      the Netscape HTTP Cookie Specification as set by including a Set-
Cookie
      HTTP Header!</BODY></HTML>";
   }
   else
   {
      $reply = "<HTML><TITLE>Bad Cookie</TITLE><BODY>Sorry, Your
browser
```

```
      doesn't appear to support the cookie protocol. If you believe you
have
      gotten this message in error... Don't eat cookies</BODY></HTML>";
   }
   print $reply;
}
else
{
   print "HTTP/1.0 302 Moved Temporarily\n\r Location: $myLoc\n\rSet-
Cookie: Cookie=Test\n\r\n\r
   <HTML><BODY>BYE-BYE</BODY></HTML>";
}
```

---

**NO Patch For Ignorance.**