Browser Identification for web applications

Abstract

Browser Identification is not a new concept. With the focus having shifted to desktops from networks and servers, a topic such as remote *browser identification* needs to be revisited.

Browsers identify themselves to web servers in the USER_AGENT header field that is contained in requests sent to the server. Almost every release of browsers contains sloppy code that allows malicious servers or attackers to compromise user privacy and security.

The header that normally identifies a user's web browser tells such servers exactly which attacks to use. Obfuscating the information contained in the USER_AGENT header field reduces the likelihood of browser-related attacks.

There are other methods of analysis and evaluation that help in accurately identifying browsers. Knowing about these methods is necessary for two reasons:

- Increase awareness of browser-related attacks among desktop users.
- Assist security consultants to factor in browser-related information when working on web application security testing assignments.

This paper outlines techniques that allow users to determine client browser types remotely.

Shreeraj Shah
Founder & Director, Net Square
Co-Author: "Web Hacking: Attacks and Defense" (Addison Wesley, 2002)
and published several advisories on security flaws.

n e t - s q u a r e
http://www.net-square.com
shreeraj@net-square.com
[blog] http://shreeraj.blogspot.com

Table of Contents

Acknowledgement

Lyra Fernandes for her help on documentation.

Problem Domain

One of the most critical problems that web applications face is to identify and validate incoming requests from clients over HTTP. Web applications are designed to serve web browsers like Mozilla, IE, Opera et cetera. But web applications are attacked by automated tools which, unlike browsers do not send exact requests. Identifying a set of requests by browsers and their attributes will go a long way in protecting web applications from these attack tools. The problem is twofold – one, to identify whether the sent request is generated by a browser, and two, the browser that the sent request originated from.

Solution

The solution to this problem can be derived from different approaches and methods:

1. Understanding the behavior of various browsers and making decisions based on this behaviour.
2. Forcing browsers to generate requests and send back to servers, a characteristic that cannot be generated by automated tools.
3. Forcing browser engines like JavaScripts to generate requests and send them it to the server. These requests or information may differ from browser to browser.

The objective of this paper is to analyze each of the above methods. This paper is much like an academic exercise but it is possible to implement solutions using dynamic pages (asp, php, etc.), ISAPI or HttpModules depending on the need. A developer can build application plug-ins to achieve this objective.

Approach 1 – Order Value Analysis (OV)

Each incoming HTTP request contains a set of header values in a specific order. These header values are static. Let us look at simple GET requests made by different browsers. For research purposes, the browsers used are Internet Explorer 5 (referred to as IE throughout this paper), Opera7.0, Mozilla 5.0.

IE

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
```

Opera

```
GET / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000) Opera 7.0
[en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

Mozilla

```
GET / HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,im
age/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Observations

a.) The order of HTTP header fields is different. This can be used as a decision-making parameter. For example, the “Host” directive is placed differently in all of

them. In some cases, the header field “User-Agent” is being sent before the header field “Host”. Important and subtle distinctions that need to be noted.

b.) Static values sent by each browser for the same directive is different. For example, the “Accept” directive. Take a look at the screenshot below:

IE - Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

Opera - Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1

Mozilla - Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1

By doing proper OV (Order-Value) analysis we can identify the browser sending requests to the server. Browsers can be validated using this technique. Opening a simple TCP connection to send a similar request to the server can fool the application since the request is identical. In the next sections, we take a look at how greater hurdles can be created for automation tools.

Approach 2 – Forcing the browser to generate “GET” request

We need to find a technique which would force browsers to send back GET requests when some web page gets loaded in the browser. This will make sure that the client making the requests to the web application is a proper web browser. In other words, we need to verify that a TCP client like netcat is not generating these requests. This puts hurdles in the path of an attacker. The same GET request which is self-generated by the browser may have a different OV structure that would essentially help in identifying the remote browser as well.

We can send the following lines via an HTTP tag. This will force the browser to send *HTTP GET requests* back to the server.

Example HTML:

```
---------- source of src.html ----------
<HTML>
  <HEAD></HEAD>
  <IMG SRC=src.jpg width=10 height=10 alt="secret" border=0 vspace=0>
</HTML>
```

NOTE: OV parameters that can be used for browser identification have been shown in a different color than the rest of the fields.

Now let us analyze various responses that we receive. The above HTML tag will force browsers to send GET requests for “src.jpg”. Here is the list of requests from our set of browsers:

IE:

```
GET /src.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
Cookie: ASPSESSIONIDSSATQBSD=BLFLIFGCKCHEGNPFNIKEBHCA
```

```
GET /src.jpg HTTP/1.1
Accept: */*
Referer: http://192.168.7.60/src.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
Cookie: ASPSESSIONIDSSATQBSD=BLFLIFGCKCHEGNPFNIKEBHCA
```

Opera:

```
GET /src.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-
xbitmap, */*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Cookie: ASPSESSIONIDSSATQBSD=CLFLIFGCFOBOKONAAEECCOFB;
magicnum=14305
Cookie2: $Version="1"
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

```
GET /src.jpg HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-
xbitmap, */*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://192.168.7.60/src.html
Cookie: ASPSESSIONIDSSATQBSD=CLFLIFGCFOBOKONAAEECCOFB;
Cookie2: $Version="1"
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

Mozilla:

```
GET /src.html HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,tex
t/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: ASPSESSIONIDSSATQBSD=DLFLIFGCDHKLJOJAJNIGIHDO
```

```
GET /src.jpg HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept: image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://192.168.7.60/src.html
Cookie: ASPSESSIONIDSSATQBSD=DLFLIFGCDHKLJOJAJNIGIHDO
```

Some of the browsers send a "*Referer*" directive, for others – since this request is for a "jpg" file – the "Accept" directive changes.

Approach 3 – Caching technique using "HTTP-EQUIV"

By using HTTP equiv one can setup caching instruction for the resource. This will force browser to send another HTTP GET request with some added directives. Following code can be embedded in HTML page.

```
---------- source of httpeqca1.html ------------
<html>
  <head>
    <title></title>
    <META HTTP-EQUIV="expires" CONTENT="0">
  </head>
  <body></body>
</html>
```

The above tag will set up an *expires* value for caching = 0. This means that every request must ask for "If-Modified" and "ETag" directives. Shown below are browser-generated sets of requests.

IE

1.

```
GET /httpeqca1.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
```

2.

```
GET /httpeqca1.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 26 Apr 2004 10:32:41 GMT
If-None-Match: "60cad8cd792bc41:977"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
```

Mozilla

1.

```
GET /httpeqca1.html HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

2.

```
GET /httpeqca1.html HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Mon, 26 Apr 2004 10:32:41 GMT
If-None-Match: "60cad8cd792bc41:977"
```

Opera

1.

```
GET /httpeqca1.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Connection: Keep-Alive
```

2.

```
GET /httpeqca1.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
If-Modified-Since: Mon, 26 Apr 2004 10:32:41 GMT
If-None-Match: "60cad8cd792bc41:977"
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

A-ha! We have here a new wealth of OV information – the order and values of the "If-Modified-Since" and "If-None-Match" directives for different browsers. This approach can help in identifying and validating the browser.

Approach 4 – Loading Cookies from HTML

It is possible to load cookie values from an HTML page. We can add *META* tags to set up cookies. If we set up two cookies, one after another in quick succession and observe responses to browser requests, we can identify browsers on the basis of these results. Here is a page which can load a cookie into the browser:

```
----- Source of cookie.html -----
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Set-Cookie" CONTENT="cook1=1">
    <META HTTP-EQUIV="Set-Cookie" CONTENT="cook2=2">
  </HEAD>
  <BODY></BODY>
</HTML>
```

Here is the list of requests and responses from our set of browsers.

Mozilla

```
GET /cookie.html HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

```
GET / HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: cook2=2; cook1=1
```

IE

```
GET /cookie.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
```

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
Cookie: cook1=1; cook2=2
```

Opera

```
GET /cookie.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Connection: Keep-Alive
```

```
GET /test HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Cookie: cook1=1; cook2=2
Cookie2: $Version="1"
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

Observe closely the responses from each of the browsers for different requests. The OV for the "Cookie" directive for each of these browsers is different. Note also how the order of *cookie* value gets changed as well – an effective parameter to factor in when identifying browsers.

Approach 5 – Forcing browsers to send "POST" request

We have seen how we can force browsers to send back GET requests using the "SRC" HTML tag. Is it possible to tell a browser to send back a "POST" request? The answer is YES! Here is the way we can do so: Include this javascript code snippet in an HTML page.

```
------- source for jspost.html -------
<HTML>
  <BODY>
    <form name="order" action="./cool" method="POST">
      <input type="hidden" name=cool value="hi">
    </form>
      <script type="text/javascript" language="JavaScript">
        document.order.submit();
      </script>
  </BODY>
</HTML>
```

Here is the list of request we got from browsers:

Internet Explorer:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
```

```
POST /cool HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Referer: http://192.168.7.60/jspost.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Content-Length: 7
Connection: Keep-Alive

cool=hi
```

Mozilla

```
GET / HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

```
POST /cool HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://192.168.7.60/jspost.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

cool=hi
```

Opera

```
GET / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

```
POST /cool HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000)
Opera 7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://192.168.7.60/jspost.html
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
Content-type: application/x-www-form-urlencoded
Content-length: 7

cool=hi
```

Bingo! We now have the POST request back. Again, as with GET, you can see a lot of values for the OV have changed. New HTTP header values like “Content-type” and “Content-length” are evident in the POST request. These OVs can be analyzed to determine browsers.

Approach 6 – Invoking JavaSscript function and get as “cookie”

It is possible to invoke a JavaScript function such as Date(). Here is a list of requests generated by browsers with Date functions. We force the script to run the function and set it as cookie so we obtain system date and time values in the header.

IE

```
GET /98463754 HTTP/1.1
Accept: */*
Referer: http://192.168.7.60/dget.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.7.60
Connection: Keep-Alive
Cookie: date=Tue May 4 15:34:44 UTC+0530 2004
```

Mozilla

```
GET /823671668 HTTP/1.1
Host: 192.168.7.60
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.6)
Gecko/20040113
Accept: image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://192.168.7.60/dget.html
Cookie: date=Tue May 04 2004 15:35:39 GMT+0530 (India Standard
```

Opera

```
GET /505630098 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows 2000) Opera
7.0  [en]
Host: 192.168.7.60
Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap,
*/*;q=0.1
Accept-Language: en
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://192.168.7.60/dget.html
Cookie: date=Tue, 04 May 2004 15:36:18 GMT+0530
Cookie2: $Version="1"
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

As you can see, for each of these browsers, the date formats are different – another OV parameter for identifying browser types.

Conclusion:

The browser identification techniques outlined in the paper have high accuracy levels, having evolved over a span of time.

Of late, naive desktop users have been the favourite target of attackers. Attackers are banking on users such as these to visit a web site or click a link to initiate attacks.

Presenting the techniques for browser identification in this paper is included more so for security consultants, who might find it worthwhile to their expand their repertoire of attack-and-penetration tests to also include tests for browser-related attacks. At the same time developers can build a defense using some of these techniques.