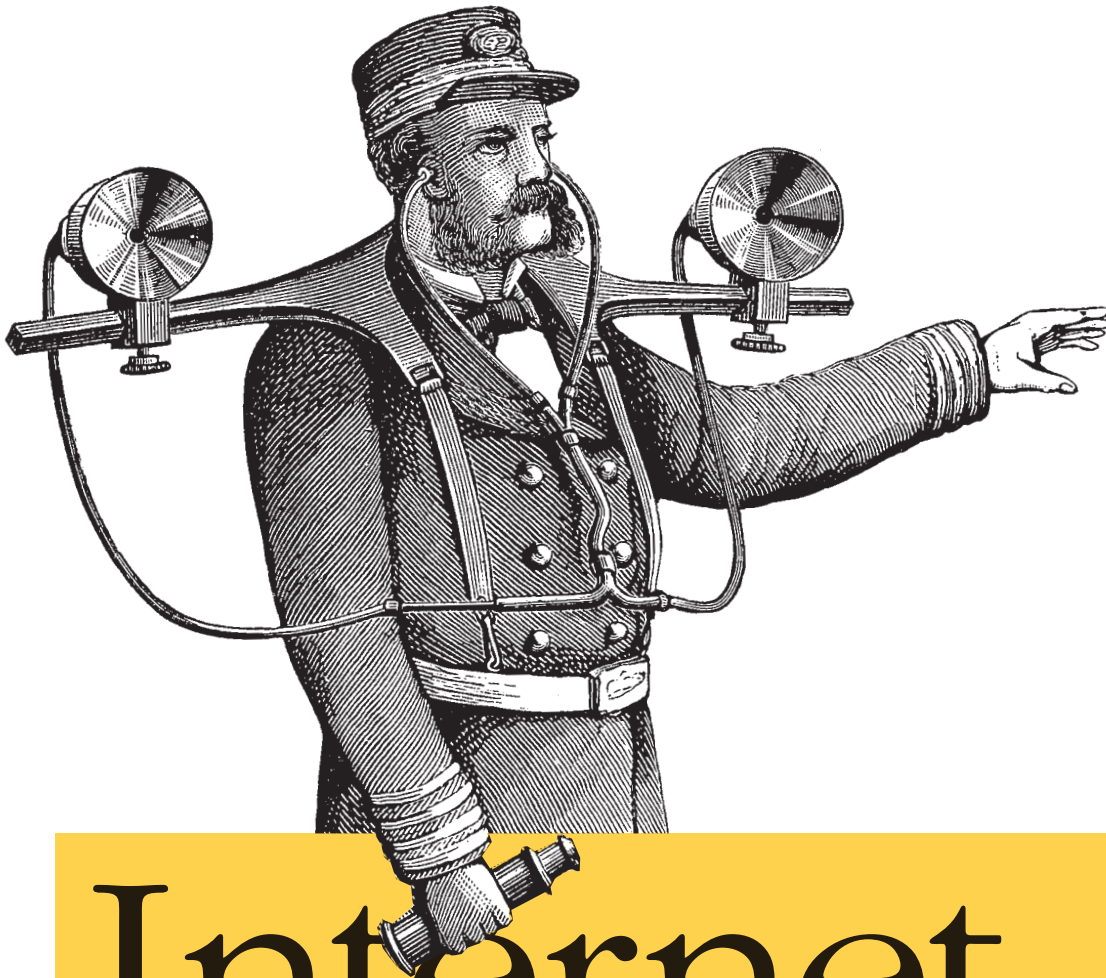


Using Digital Evidence to Solve Computer Crime



Internet Forensics

O'REILLY®

Robert Jones

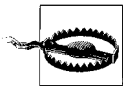
Obfuscation

The Achilles' heel of any Internet con artist is the web site they use to trick their victims. In order for the scam to function, victims have to be able to access a real site at a defined location on the Internet. But revealing that address opens the door for investigators, leading to their sites being shut down and perhaps to their true identities being discovered.

The bad guys are very aware of the problem and go to great lengths to disguise, or obfuscate, their real addresses in the vain hope that investigators will be fooled or become frustrated and give up the pursuit.

On top of that, spam-blocking software is making it increasingly difficult for their emails to get through to our mailboxes. Anything that can disguise an address and avoid it being added to a spam blacklist will extend the life of a scam—so spammers will use every trick in the book.

It's a bit like an arms race, with pressure from our side forcing them to innovate and come up with new tricks. Fortunately for us, implicit in any form of obfuscation is the fact that browsers must be able to reveal the true URL in order to use it. If the browser can do it, so can we. This chapter covers a variety of tricks, some of them quite elegant, that scammers use to throw us off the scent of their trail.



The developers of Internet browsers are continually updating their software to address security exploits, including some of the tricks described here. As a result, with any given browser, some tricks will work and others will not. In due course, you can expect that many will be completely blocked. But these things have a way of reappearing in different contexts, so I will describe the complete menagerie.

Anatomy of a URL

Here are a few examples of URLs that illustrate the problem:

- `http://www.craic.com`
- `http://208.12.16.5`
- `http://%77%77%77%2e%63%72%61%69%63%2e%63%6f%6d`
- `http://www.oreilly.com@www.craic.com`

All of these take you to my web site, but only the first one is recognizable by the casual user. Most of these variants use the more arcane features of the URL specification, so I will start with a brief review of that. The general syntax of a URL is as follows:

`<protocol>://<user>:<password>@<host>:<port>/<url-path>`

This can be simplified to produce something that looks almost familiar:

`<protocol>://<host>/<url-path>`

`<protocol>`

This notation refers to the network protocol being invoked to transfer data back and forth. This is usually the hypertext transfer protocol (http) but other options include https, ftp, file, and mailto.

`<host>`

The address of the web server, represented as a *fully qualified domain name* (FQDN), such `www.craic.com`, or a numeric IP address, such as `208.12.16.5`.

`<url-path>`

The path to a specific file or directory on that web server.

`<port>`

This allows you specify the TCP/IP port to use in the http transaction. The default port is 80, but you sometimes see other ports specified, such as 8080.

`<username>` and `<password>`

These are rarely seen in normal URLs. When you visit a web site that has restricted access you usually authenticate yourself via a pop-up window. You can enter the same information in the URL if you want to, but this is a very bad idea because your password is in plain view. In fact the only people who use this mechanism are the bad guys trying to con us, as you shall see shortly.

The ultimate reference for this syntax is RFC1738, “Uniform Resource Locators (URL),” issued by the Network Working Group in 1994 and written by Berners-Lee, Masinter & McCahill (<http://www.w3.org/Addressing/rfc1738.txt>).

Encoding Characters in URLs

Alongside the syntax are the *encodings* that can be applied to the different components. Certain characters such as &, ?, and = have special meanings in a URL string. Including these in the name of a file on a web site could have unwanted consequences when interpreted by a web server in the context of a URL.

By way of an example, consider the slightly contrived example of an HTML file with the name *test?key=value.html*. In the form of a URL, it looks like this:

```
http://www.craic.com/test?key=value.html
```

The web server takes this string at face value and tries to execute a CGI script called *test*, setting the parameter *key* to *value.html*. The server returns an error because the script *test* does not exist. You get around this by encoding the special characters in hexadecimal. The web server ignores these when parsing the URL, converting them only when it tries to retrieve the file.

Hexadecimal codes are two-character strings and are denoted within a URL by a preceding % character. You can see the entire hexadecimal character set on a Unix system with the command `man ascii`. In the previous example, ? is encoded as %3f and = as %3d. When you type this form of the URL into the browser, you see the intended web page.

```
http://www.craic.com/test%3fkey%3dvalue.html
```

Any other character in the URL path or hostname can be encoded in hexadecimal. The one you will be most familiar with is the space character, encoded as %20. A number of web browsers will encode spaces automatically if you include them in your URL. Spaces can also be replaced by + characters.

This mechanism is part of the URL specification, so web servers are built to handle them. This feature allows you to encode not just the special characters but essentially entire URLs in hexadecimal and have them function normally. Hence the URL for my web site can be represented as:

```
http://%77%77%77%2e%63%72%61%69%63%2e%63%6f%6d
```

Decoding a hexadecimal URL back to ASCII is tedious in the extreme, so Example 4-1 provides a simple Perl script that does the job for you. Example 4-2 allows you to encode your ASCII text as hexadecimal.

Example 4-1. decode_hex_url.pl

```
#!/usr/bin/perl -w
die "Usage: $0 <hex encoded URL>" unless @ARGV == 1;
$ARGV[0] =~ s/\%(..)/chr hex $1/ge;
print $ARGV[0] . "\n";
```

Example 4-2. `encode_hex_url.pl`

```
#!/usr/bin/perl -w
die "Usage: $0 <ASCII URL>" unless @ARGV == 1;
for(my $i=0; $i < length $ARGV[0]; $i++) {
    my $c = substr($ARGV[0], $i, 1);
    printf "%02lx", ord $c;
}
print "\n";
```

Here is a real example using a hybrid of ASCII and hexadecimal to make you think it is a legitimate URL at a major bank. It's a long URL so I've had to split it into two lines:

```
http://web.da-us.citibank.com%2E%75%73%65%72%73%65%74%2E%6E
%65%74:%34%39%30%33/%63/%69%6E%64%65%78%2E%68%74%6D
```

Translated back to ASCII, it reveals that the bank's domain is simply part of the hostname of a totally different server:

```
http://web.da-us.citibank.com.userset.net:4903/c/index.htm
```

International Domain Names

Historically, domain names have only been able to include letters from the English alphabet, numbers, and dashes. This has posed a problem for companies in non-English-speaking countries that wanted a domain name that matched their brand as written in Arabic, Chinese, and so forth. The workaround to this is called *International Domain Names* (IDN), and it involves encoding non-English characters, represented in *Unicode*, as basic ASCII strings. This encoding is called *punycode*. The idea is that the existing machinery of the Internet will continue to use the limited character set but that web browsers would decode punycode entities into their real representation. For example, the domain `bücher.ch`, with a single non-ASCII character, would be represented as `xn--bcher-kva.ch`. It's an ugly syntax, but that would normally be hidden from the user.

There is a lot of interest in IDN at the moment, and most of the major browsers do support it. But this new functionality brings with an opportunity for those who want to impersonate the URLs of other companies. Unicode is able to represent essentially every character in every language used in the world today, and then some. Many of those codes can be handled by punycode. Among them are equivalents to standard ASCII characters, which can be used to trick a user into thinking they are going to one site when in fact they are taken to something quite different. For example, the Unicode character called "Cyrillic Small Letter A" looks exactly like the ASCII lowercase `a` when displayed in a browser. This is called a *homograph*, but because it is a non-ASCII character, it can only be represented in an encoded IDN. Eric Johanson and colleagues in The Shmoo Group (<http://www.shmoo.com>) realized this and have published the exploit in order to publicize the problem.

They encoded the string `paypal.com` in punycode, replacing the first `a` with the Cyrillic character. This resulted in the string `xn--pypal-4ve.com`, a new domain that they proceeded to register. Anyone entering `http://www.xn--pypal-4ve.com` into an IDN-enabled browser will see it translated to `http://www.paypal.com` but the returned page comes from the first domain.

This is a very clever exploit that has some serious implications for the success of IDNs. It has yet to turn up in a real phishing attempt, but it received quite a lot of publicity following its publication. In response, new downloads of the Firefox and Mozilla browsers have IDN support turned off by default. One solution to this would be to remove support for specific encoded homographic characters in browsers and to prevent domain names that contain them being registered. But that will require significant cooperation from domain registrars, which may be difficult to obtain.

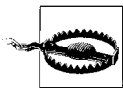
IP Addresses in URLs

We expect a URL to include the hostname of a web server but we can just as easily use the numeric IP address in its place. `http://208.12.16.5` and `http://www.craic.com` are completely equivalent. But most people don't remember the IP address of their own computer, let alone one for eBay or Citibank. Most people tend to assume that an IP address is valid, whereas a false hostname is more likely to arouse suspicion. Scammers exploit this and often use IP addresses in their URLs.

There is a second, perhaps more valuable, benefit to this approach. You can set up an account with an ISP, be assigned an IP address, and set up a web server without having registered a domain name. It makes it harder for people to find you, but because you are including the URL in your spam, that is not a problem. In fact, it is a significant advantage.

Here are a few examples:

- `http://202.87.128.138/sys/index.php`
- `http://211.250.185.100/~bookmaul/paypal/login.html`
- `http://218.244.98.8/wamu`



URLs with IP addresses may not work properly if the web server manages several virtual hosts. The hostname allows the server to direct you to the correct site, but the IP address is ambiguous, and you will see the first site in the server configuration file that matches that address.

Encoding the IP Address

The IP address alone is not a great disguise, so it is not surprising to see another layer of deception being added by encoding the address in some way.

The easiest approach is to encode the characters in the address in hexadecimal as we did earlier. In this way `http://208.12.16.5` becomes `http://%32%30%38%2e%31%32%2e%31%36%2e%35`.

An interesting alternative is to change the representation of the IP address itself. You can think of a dotted-quad address as a number in base 256, in which the four parts become four successive digits. We can convert this to standard decimal number. If the address has the form A.B.C.D, then the decimal form is calculated thus:

$$A*(256**3) + B*(256**2) + C*256 + D$$

So `208.12.16.5` becomes:

$$(208 * 16777216) + (12 * 65536) + (16 * 256) + 5 = 3490451461$$

Give this a try: `http://3490451461`

You don't see this very often in practice, probably because it doesn't work in Internet Explorer, but it does work in Firefox on Mac OS X.

Finally, if you want to get really cryptic, you can encode each *part* of a dotted-quad address in *octal*, precede the numbers with a zero, and separate those with periods. In this form, the address `208.12.16.5` becomes `0320.0014.0020.0005`.

Example 4-3 provides a Perl script to encode a numeric IP address in octal, and the script in Example 4-4 performs the reverse transformation. It is rare to find octal URLs in spam emails, but they do occur and are functional in Safari on Mac OS X.

Example 4-3. encode_octal_url.pl

```
#!/usr/bin/perl -w
die "Usage: $0 <dotted quad IP addr>\n" unless @ARGV == 1;
my @words = ();
foreach my $word (split /\./, $ARGV[0]) {
    push @words, sprintf "%03lo", $word;
}
printf "%s\n", join '.', @words;
```

Example 4-4. decode_octal_url.pl

```
#!/usr/bin/perl -w
die "Usage: $0 <octal encoded URL>\n" unless @ARGV == 1;
$ARGV[0] =~ s/(O\d\d\d)/oct $1/ge;
print $ARGV[0] . "\n";
```

Username in URLs

The encodings described above can be fairly effective at disguising the underlying hostname of a web site, but they don't look like regular URLs and that alone can attract suspicion. A far more convincing URL is something like:

`http://www.oreilly.com@www.craic.com/`

Table 4-1. Base64 character set (continued)

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	Pad	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The encoding allows 6 bits of input data to be represented by a single ASCII character. A byte has 8 bits, so the encoding takes 3-byte chunks of data, which is 24 bits, and encodes it as 4 ASCII characters. As you can see, this is not a compression scheme. You commonly compress a file first and then encode its binary data using Base64.

Manually decoding the output would be extremely tedious. One way to handle this is to copy the encoded text, and nothing else, into a file and pass it to the Unix program `openssl`.

```
% openssl enc -d -a -in your_file
```

An alternative is to install the `MIME::Base64` Perl module on your system and then use this Perl one-liner to decode it.

```
% perl -MMIME::Base64 -ne 'print decode_base64($_)' < your_file
```

The example given previously decodes to this simple web page:

```
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Dear e-gold user !</p>
<p>Our system has undergone to serious preventive maintenance,
please, check up functioning your e-gold account.</p>
<p>The e-gold site is at: </p>
<p><a href="http://www.e-gold.com@reynsan.netfirms.com">
http://www.e-gold.com</a></p>
<p>This is automatic email.<br>
    Do not reply to this email.</p>
</body>
</html>
```

This is simply the HTML code for the text that was displayed in the mail client. The difference is that you can see the real target for the URL in that message: `http://www.e-gold.com@reynsan.netfirms.com`. Well, not quite. This URL is actually the one I used as an example in the earlier section *Usernames in URLs*, with 140 %01 padding characters. This character is a non-printing ASCII character, so when you view the decoded output in `more`, you don't see them. Open up the output in `emacs` and they are visible as 140 `Ctrl-A` characters.

If that all seems unnecessarily complex, remember that one reason for the disguise is to defeat spam-filtering software. Unless that software can decode Base64 to get at the real text then it can't tell if this is a legitimate message. The same motivation leads some spammers to make images containing the text of the message, perhaps captured from a screen dump. The email messages may contain a URL to an image on a remote server or may include the image as a block of encoded text within the message. The images are placed within an anchor tag, so that you can click anywhere on the image and go to the target URL.

This target URL was taken from a message that supposedly came from a bank. Just to make things more of a challenge that URL was encoded:

```
http://%32%32%31%2E%31%38%34%2E%39%32%2E%31%36%39%34%39%30%33%63%69%74%69%6E%64%65%78%2E%68%74%6D
```

Translating that yields a numeric IP address and a nonstandard port number:

```
http://221.184.92.169:4903/cit/index.htm
```

This is a good example of the multiple layers of deception that profession scammers will employ to make it difficult for spam filters and people like us who want to reveal them.

Similar Domain Names

A simple and widely applied form of trickery for phishing attempts is to use domain names that look very similar to the original. Here are a few of *many* examples:

Real domain name	Fake domain name
<i>citibank.com</i>	<i>mycitibank.org</i>
<i>citizensbank.com</i>	<i>citizensbankonline.com</i>
<i>usbank.com</i>	<i>ussbank.net</i>
<i>firstusa.com</i>	<i>firstusaonline.biz</i>
<i>washingtonmutual.com</i>	<i>washingtonmutual.com</i>

Companies such as PayPal and eBay try and protect themselves by registering a range of domain names similar to their primary domain. But a creative scammer will

always be able to come up with some new twist on the name that has not yet been registered. Netcraft's web-based DNS search tools (<http://searchdns.netcraft.com>) can show the scale of the problem. Select the site contains option, enter a name such as paypal, and see how many web servers are reported that match. Some of these are legitimate but a lot of them look very dubious.

Making a Form Look Like a URL

In most web browsers and email clients, when you move your mouse over a hyperlink, you will see the target URL displayed in the status bar at the bottom of the window. This can be a real giveaway for an obfuscated URL, so some effort has been applied to prevent this happening.

One elegant approach, if you can use that term in the context of Internet fraud, is to replace a simple hyperlink with an HTML form that contains just a single SUBMIT button. In most cases this would stick out like a sore thumb, but through the use of STYLE attributes you can make this button look exactly like a regular <A> HTML anchor.

Placing your mouse over the fake anchor results in no message in the status bar. Here is a code snippet that shows this in action:

```
<form action="http://www.craic.com">
  <input type="submit" value="http://www.craic.com"
  style="font-family: times; font-size: 12pt; color: blue;
  text-decoration: underline; border-width: 0pt; padding: 0pt;
  background-color: transparent;" >
</form>
```

The way you can tell the difference between a regular hyperlink and a modified submit button is by the cursor when you mouse over it. This changes to the familiar “hand” cursor when over a regular hyperlink, while staying as a basic pointer when placed over the submit button.

As with many of these tricks, different browsers treat them differently. This one works as the authors intended in Firefox on Linux and Internet Explorer on Mac OS X, but it still appears as a regular SUBMIT button in Safari on a Mac OS X system.

Bait and Switch—URL Redirection

One alternative to disguising the URL of a site is to post the address of a second site and have that redirect any traffic it receives to the target address. You think you're going to site A but you end up at site B.

This takes some effort to set up as the owner has to have control of one or more of these proxy web sites. The benefit to them is that it hides the identity of their main web site, the one that actually sells a product or steals your identity. Nowhere in any

of the junk emails does that address occur. As one proxy site gets exposed, and its address is added to spam blacklists, it is easy to set up another proxy site and to be back in business. The effort and cost of setting up these sites is minimal and much easier than having to move the primary site from one address to another.

There are two main ways to accomplish this sort of redirection. You can add a special tag to a web page on the proxy site that tells your browser to go to the target. Or, you can add a line to the web server configuration file that intercepts the request for a specific page and tells the browser to fetch it from the target location.

Page-Based Redirection

The easiest way to implement redirection is to create a web page and add a meta tag to the HEAD section of the document. meta tags are used for various purposes, such as adding keywords for search engines. One class of these uses a `http-equiv` attribute, which adds its content to the HTTP headers that are sent back to the browser immediately before the content of the page. By setting the `http-equiv` attribute to `refresh`, you can tell the browser to load a second page at some interval after loading this page. This is sometimes used to reload dynamic web pages or to create a simple slideshow effect. But we can redirect a browser to a different page immediately if we set the delay to zero seconds and include the target URL in the content attribute of the tag. Here is an example that tells the browser to immediately start loading the target URL:

```
<meta http-equiv="refresh" content="0; URL=http://www.craic.com">
```

If we changed 0 to, say, 5, then the current page would be displayed for 5 seconds before the target began to load.

Under normal circumstances you won't even see this first web page. But if things are running slowly for some reason, then you might notice it for a second or two before the target appears. The benefit of this approach is its simplicity. The owner only has to copy a web page onto the proxy site, and it will work immediately.

Server-Based Redirection

The better way to redirect users is to modify the configuration file for the web server. Almost 70% of sites on the Internet run the Apache web server, so I will only consider that software here. The operation of the server is configured in the file `httpd.conf`. Server-based redirection can be achieved in several different ways. One of the easiest is to include a one-line `Redirect` directive in the file and restart the server. The format of this is simply:

```
Redirect <old url> <new url>
```

The old URL is the local path to the HTML document on this proxy server and the new URL is the complete URL of the page on the target site. Here is an example that would take any request for *redirect.html* and redirect it to O'Reilly's home page:

```
Redirect /redirect.html http://www.oreilly.com
```

Note that I do not need to actually have a file called *redirect.html* because the web server intercepts the request before trying to retrieve the page. It sends an HTTP response back to the browser, telling it the requested page is no longer located there and giving it the new address. The browser then sends a second request to the target server for the real page. The user is none the wiser to any of this and simply sees the target page appear. Because the only page downloaded to the browser is the intended one, there is no risk of the dummy page appearing.

The downside of this approach is that the owner must either have access to the Apache configuration file or must persuade the server administrator to make the changes on their behalf. Either way, it demands a higher level of sophistication on the part of the people responsible for the site.

A second approach that is widely used by phishing web sites is to generate a web page from a server-side script, typically written in Perl or PHP. That script generates the HTTP headers for the page before it outputs the contents of the page itself. Including a Location header will direct the browser to fetch the specified URL instead of displaying the following content. For example, the following header would redirect the browser to the O'Reilly web site:

```
Location: http://www.oreilly.com
```

Determining the Mechanism

The observant user will know when a page has been redirected because the URL in the browser address bar will not be what you expected. You can go one step further and determine which of the two redirection methods was actually used in any given case.

The way you do this is by looking at the HTTP headers that are returned to your browser by the initial request. I talk about headers and how to access them in Chapter 6, but here is a taste of how useful they can be.

With page-based redirection, the browser fetches the requested page from the proxy site, and then acts upon the refresh directive, fetching the target page. As far as the proxy server is concerned, this is a regular http transaction and it sends back a numeric response code of 200. In plain language, the server is telling the browser that everything is OK and here is the content that you requested. Here are the edited headers that are returned by the program *wget* when given a URL that redirects using a meta tag:

```
HTTP request sent, awaiting response...
 1 HTTP/1.1 200 OK
 [...]
```

With server-based redirection, the initial page is never sent to the browser. Instead, the server responds with a set of headers that include a response code of 302. All codes in the 300 series signify server redirection in slightly different flavors. 302 stands for “Moved Temporarily,” and it’s the one you will see most frequently. It also supplies a Location header that contains the target URL. So this type of redirection results in two sets of headers. The first comes from the proxy, informing the browser of the redirection. The second comes from the target server, giving the browser an OK response followed by the content. Those headers look like this:

```
HTTP request sent, awaiting response...
 1 HTTP/1.1 302 Found
 [...]
 4 Location: http://www.craic.com
 [...]
Location: http://www.craic.com [following]
 [...]
HTTP request sent, awaiting response...
 1 HTTP/1.1 200 OK
```

This pattern of headers is exactly what I see when accessing the URLs contained in the many emails that I have received recently, trying to sell me replica Rolex watches. These point to a variety of web sites with cryptic names, all of which redirect me to `online-replica-store.com`.

Redirection via eBay

There is a third way to handle redirection that, frankly, is beyond belief. The following URL was contained in an email that appeared to be from eBay, asking me to update my credit card number.

```
http://cgi4.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain&
DomainUrl=http%3A%2F%2F%32%31%31%2E%31%37%32%2E%39%36%2E%37%2F
UpdateCenter%2FLogin%2F%3FMfcISAPISession%3DAAJbaQqzeHAAeMWZlH
h1WXS2A1BXVShqAhQRfhgTDrferHCURstpAisNRqAhQRfhgTDrferHCURstpA
sNRpAisNRqAhQRfhgTDrferHCUQRfqzeHAAeMWZlHh1Wxh
```

We know what’s going on here. It appears to be from eBay, but it’s a safe bet that hexadecimal encoded text contains a @ character with the real URL to the right of it. Here is the string after decoding:

```
http://cgi4.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain&
DomainUrl=http://211.172.96.7/UpdateCenter/Login/?MfcISAPISession=
AAJbaQqzeHAAeMWZlHh1WXS2A1BXVShqAhQRfhgTDrferHCURstpAisNRqAhQRfhgT
DrferHCURstpAisNRpAisNRqAhQRfhgTDrferHCUQRfqzeHAAeMWZlHh1Wxh
```

There is the target host on the second line: `211.172.96.7`. But look at what is *missing*. There is no @ character, so this is not using the username trick. This is a genuine eBay URL! The scammer has figured out that eBay has a script that will redirect visitors to different sites, presumably within the eBay domain. But they have dropped the ball in a big way by allowing it to redirect you to *any* site that is supplied in the

DomainURL parameter. So the scammer has eBay handle the redirection that takes a visitor to the fake site. You can feed any URL to the script and it works fine.

```
http://cgi4.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain&
DomainUrl=http://www.cnn.com
```

Can this be true? Can one of the leading targets for phishing attacks have left open a security hole so large that you could drive a whole truck full of secondhand collectibles through it?

There are two things that you have to do when you come across situations like this. The first is to laugh out loud. The second is to email the company involved and warn them about the problem, which I did. It turns out that in order to email eBay's security people, you first have to open an account on eBay! I provided a detailed description of the issue and sent it in. So far the only responses I have received are an automated acknowledgement with advice on how to spot phishing attempts, and a "Welcome to eBay!" message.

Perhaps the script has been set up like this on purpose, as a honeypot with which to entice the scammers. By looking at the URLs being fed to the script, eBay could quickly identify the fake sites and shut them down. But that cannot have been the case here because the redirect and the fake site both worked fine. At the very least, that script should have spotted URLs that were outside the company and returned a page warning that this was a phishing attempt. The outlook for scammers looks bright if vulnerabilities like this go unnoticed. At the time of writing, this redirection still works.

JavaScript

In principle, *JavaScript* applications, embedded in web pages, are a great way for the bad guys to cover their tracks. In principle, you could write scripts to prevent someone from viewing the source HTML for a page and to manipulate the URL displayed in the status bar and browser history. Poke around the Web and you will find lots of example scripts claiming to do these and other feats of engineering. The problem is that most of them do not work.

While most browsers support JavaScript, and most have that support turned on by default, recent versions tend to disable functions that interfere with the way the browser functions. You can program image rollovers and form validation just fine, but try anything else and it may not work. These restrictions are a direct response to people trying to trick users or exploit security holes that give them access to files and so on.

Some of the tricks still work in certain browsers. Here is a snippet of HTML that displays a false URL in the name of the link and uses JavaScript to rewrite the status bar with the same URL.

```
<a href="http://www.craic.com" onMouseOver="window.status =  
'http://www.oreilly.com'; return true;">http://www.oreilly.com</a>
```

This one still works in Internet Explorer 6.0 on Windows. Safari on Mac OS X shows the real target, as does Firefox on Mac OS X, unless you specifically change your preferences to allow JavaScript to mess with the status bar.

These scripts can only work if run inside a browser that can interpret JavaScript. In most of our explorations, we will be using a command-line browser that simply fetches the HTML for us. So these scripts, regardless of how sophisticated they may be, serve as no impediment to our explorations.

Browsers and Obfuscation

The variation in behavior between browsers when given some of these obfuscated URLs is frustrating. On the one hand, it shows that the developers of these tools are aware of the problem and are doing something about it. But on the other, they are building browsers that do not implement the accepted specification for URLs. While their design choices may help solve an immediate problem, they will also break any legitimate use of these features.

It is also clear, from the differences in behavior, that each development team is going its own way rather than working toward a common goal. On top of this, we are now seeing a plethora of add-on toolbars, notably for Internet Explorer, which can alert users to some forms of obfuscation. Here are three examples of those:

- *<http://toolbar.netcraft.com/>*
- *<http://www.earthlink.net/home/software/toolbar/>*
- *<http://pages.ebay.com/ebay/toolbar/>*

What we need is a revision of the URL specification combined with a coordinated effort among browser developers to implement that standard. We will undoubtedly lose a few features from the current specification, which will upset some people, but it would make life quite a bit harder for the scammers.