Anti-Phishing: Page Encoding

Hacker Factor Solutions
Whitepaper

Revision history:
25-March-2005: Proof of concept code developed and tested.
29-March-2005: First draft completed.
2-April-2005: Incorporated reviews and feedback from Rachael Lininger.

Hacker Factor
P.O. Box 270033
Fort Collins, CO
80527-0033
http://www.hackerfactor.com/

Table of Contents

Table of Contents 2
1 Overview 3
2 Solution Background 4
2.1 Phishing Skills 4
2.2 Free-Trade Market 4
2.3 Common Look and Feel 5
2.4 Mirroring Techniques 5
2.4.1 Identifying Internet Explorer Mirrors 6
2.4.2 Copy and Paste 6
2.4.3 Web Editing Tools 6
2.4.4 Mirror Limitations 7
3 Solution Details: Page Encoding 8
3.1 Benefits of Encoding 8
3.2 Limitations to Encoding 9
3.3 Anticipated Evolution 10
3.3.1 Short term 10
3.3.2 Mid term 11
3.3.3 Long term 11
4 About Hacker Factor 12
5 Appendix A: Sample Code 13
6 Appendix B: Example Encoding 14

1 Overview

For the last three years, Hacker Factor Solutions has been investigating phishers and phishing activities. This research focuses on three core areas:

- Profile the individuals and groups involved in phishing.
- Classify phishing attacks and attack methodologies.
- Develop anti-phishing options.

The premise driving this research is that if you know your opposition, then you know their weaknesses and can develop solutions that address these weaknesses. Due to the dynamic nature of phishing, nobody realistically expects a single solution to be completely successful, and phishing technology evolves around successful anti-phishing methods.

Hacker Factor Solutions identified a specific pattern and weakness. We believe this weakness is used by more than 80% of phishing groups and individuals. The weakness centers on the creation of phishing emails and web servers. The proposed solution, **page encoding**, encapsulates each web page in a simple encoding function, and uses JavaScript to decode the page contents. Proof-of-concept source code is provided in Appendix A. An example of page encoding appears in Appendix B.

The goal of page encoding is to change the dynamics of the phishing free-trade underground. We believe it will create a bottleneck between supply and demand for phishing scams. This, in turn, should decrease the number of phishing attacks and lower the likelihood of a successful attack.

The goal of page encoding is *not* to prevent phishing. It is only designed to deter future phishing attacks and make sites less desirable to phishers. Page encoding operates as a defense-in-depth component with other anti-phishing solutions.

This paper is released publicly for the following reasons:

- We expect adoption of this technique to significantly impact the ability of phishing groups to imitate trusted corporations.

- Silent adoption will be readily identified by phishers, and only slow the potential adoption rate by legitimate companies.

- Knowledge of this technique will not impact phishing groups' abilities to adapt around this solution. We believed that phishers would only be able to react after it is deployed, and their reaction will likely be delayed by more than 12 months.

- Open discussion of this approach, including its tradeoffs and viability, is best performed in a public forum. If this solution is widely accepted, then open discussion spurs phishing targets to incorporate this solution.

- This proposed solution might not be viable for many environments, but open discussions will assist in identifying alternatives.

2 Solution Background

The majority of phishing email content, and HTML code on phishing servers, comes from mirroring web sites. The mirroring is usually achieved with the Internet Explorer (IE) "File, Save As" menu option. Although IE is the most common mirroring tool, other browsers include "Save As" options and stand-alone mirroring tools, such as the command-line 'wget' program, exist. Mirrored web pages normally include both HTML and images.

After mirroring the web site, the phisher edits the hyperlinks in order to make a working scam web page. Phishers have learned that more accurate impersonations yield more phishing victims. If the target company's web page cannot be mirrored, then the phishers cannot make perfect copies of the target company. This, in turn, makes the company a less desirable phishing target.

2.1 Phishing Skills

While a few phishing groups employ very knowledgeable software developers, the vast majority of people involved in phishing have little or no programming skills. The general skill set of these less-technical people comprises only basic editing and configuration knowledge.

- **Edit**. They can edit an HTML document, changing URLs. They cannot create an HTML document from scratch. They cannot develop an imitation web page without starting with a template (mirror) of the target company. Templates that use style sheets and complex HTML formatting increase the modification difficulty, making complicated edits impractical for less-technical people.

- **Configure and execute**. They cannot program or edit source code. But they can compile and run applications. In some cases, they can also perform simple application configuration modifications.

A few of the professional phishing groups employ very technical software developers. But the remaining phishing groups either have no technical members, or few technical members. As such, there are a limited number of ways that phishers can acquire phishing scams:

- **Develop**. Some phishers develop servers from scratch. This requires significant skill that phishers in general do not possess. Due to the complexity, developing new servers is the least common approach. Instead, a new server is usually created by one group and then traded to (or stolen by) other phishing groups and used as a template.

- **Modify**. Using a mirroring tool, phishers can acquire a template for modification. This appears as many similar phishing servers, but with minor differences due to customization. This is the most common approach for creating a new phishing scam. Tools, such as IE's "Save As" option, simplify the acquisition process to a level that less-technical phishers can operate.

- **Trade**. Phishers that develops a new scam may trade it to multiple less-technical phishers. Trading is nearly as common as modification. This appears as multiple phishing groups using the same phishing server web pages and configuration. This type of cross-pollination makes it difficult to identify phishers from web server configurations.

2.2 Free-Trade Market

Much of the underground phishing community is centered on a free-trade market. People with different skills and knowledge exchange valuable information for other skills or knowledge. When trading, different items have different values. These examples include text from an active phishing forum:

- **Identities**. A full identity (name, address, credit card, social security, etc.) is more valuable than a partial identity. For example, three credit cards with CVV2 information trade for one full identity. ("`WILL TRADE 3 VALID CVV2 FOR 1 FULL UK including bank name and stuff OR EBAY SELLER ACCOUNT FROM UK`")

- **Mailers**. Mass mailers trade for 3-5 full identities. ("`Need PhpMailer...I will give 3 fullz for it!!!`" and "`needs php mailer send to all / I offer 5 fulls`")

- **Scams**. A "scam" (pre-packaged phishing server) costs between 4 and 10 full identities. Alternately, the provider may ask for a cut of the take.

- **Accounts**. A shell account (for anonymizing logins) can cost as much as 4 full identities.

- **Music**. A single music album (converted to MP3) costs up to 1 full identity. Although most music can be downloaded over P2P networks, P2P offers no guarantee of quality or even the right album. Thus, there is a market value for illegal music sharing.

Other items of value include mass mailing lists, compromised hosts, DNS registration, botnets, simple automation tools, computer hardware, and drugs. The most expensive items appear to be zero-day exploits, with a value in excess of $10,000US (depending on the type and scope of the exploit).

2.3 Common Look and Feel

Recognizable corporate images and branding are as important as (if not more than) the products and services offered. Branding allows customers to recognize the company. Most corporations maintain relatively static web pages: a corporate logo, consistent color scheme, and layout. But these web pages evolve over time, as evident by archives stored at The Wayback Machine <http://www.archive.org/>.

- **Fonts**. Font sizes, typefaces, and position vary. Over the course of 7 years, eBay has changed their primary font size from "unspecified" to ranges between 9pt and 12pt. Different font sizes, typefaces, and formatting changes as rapidly as monthly.

- **Positioning**. Depending on the page content, additional frames or boxes may be added or removed.

- **Content**. Text, particularly on the front-page of web sites, frequently changes. Menus are reordered, and options are added and removed. Static text becomes outdated and users notice.

Although these changes are relatively minor, frequent visitors to a web site notice when changes happen, even if they are not sure what the specific change was. (In the case of expected changes, such as stock listings or current news, nearly all users notice when content is stale.) Phishers have learned that more accurate imitations fool more victims. This motivates phishers to acquire recent mirrors for their scams. Most phishing groups use mirrors less than three months old. A few phishers use mirrors that are over a year old, but they are a minority.

2.4 Mirroring Techniques

Many techniques exist for mirroring web pages. While some methods require a high degree of technical skill, most have been simplified to automated systems and marketed toward people with little or no technical skill.

- **Easy**. The simplest mirroring systems use web browsers to copy the web pages. Internet Explorer, Netscape, Firefox, Opera, Mozilla, and Safari all offer "Save As" options for copying a web page. This mirroring process copies the HTML text, style sheets, and images.

- **Moderate**. Dedicated non-browser tools, such as 'wget', WebWhacker, and Templeton, offer more complete mirroring solutions. These tools not only mirror a single web page, but also mirror entire web sites.

- **Hard**. The most complicated mirroring approach uses command-line tools such as telnet, netcat (nc), or proprietary tools/scripts.

Many of these mirroring tools modify the actual HTML content. These modifications appear as unique residues that can be used to identify the mirroring tool.

2.4.1 Identifying Internet Explorer Mirrors

For phishing, Internet Explorer is one of the most widely used mirroring tools, used by approximately 60% of all phishing groups.[1] When IE mirrors web pages, it transforms the HTML code. These transformations appear as tell-tail signs of IE mirroring. The transformations include:

- **DOS new lines**. Unix and Windows use different new-line sequences. Unix uses a new-line character, while DOS uses both new-line and linefeed. Every mirrored line will be terminated by a new-line and linefeed. But, this may not be present in phishing email content.
- **100 characters**. Long lines are oddly wrapped with a column width of 100 characters.
- **Capital tags**. HTML tags are converted to uppercase.

While other attributes exist, these are the predominant markings set by IE. A majority of unique phishing servers and email content match the modifications left by IE.

2.4.2 Copy and Paste

A less technical approach for mirroring web pages uses the Windows clipboard. A web page displayed in Internet Explorer can be selected (control-A), copied (control-C), and pasted (control-V) into Microsoft Word. The page displayed in Word maintains much of the same formatting as the viewed web page. But saving the web page as HTML creates many significant differences:

- **Formatting styles change**. Style settings appear nearly identical on the same computer, but different computers with different fonts, screen resolutions, and even browser versions display the mirror differently than the original.
- **Fonts**. Microsoft Word strictly defines font types and sizes. For example, Word may change a web page's Arial 11pt to Veranda 8.5pt. These web pages will appear different on most other computers.
- **Backgrounds**. Background colors and images defined in style sheets and HTML are lost.
- **Images**. GIF and Jpeg images are copied without modification, but file names change.
- **JavaScript**. All scripting on the web page is lost. This includes all JavaScript, Java, and ActiveX components. For many e-commerce sites, scripting plays an important part in defining the look-and-feel; without scripting, users will notice.

2.4.3 Web Editing Tools

Web page editing tools, such as FrontPage and Dreamweaver, permit web page modification without losing format. User can import a complicated (or encoded) page and viewed it in the preview window. Content from the preview window can be copied back into HTML with almost no formatting loss. The main limitations are:

[1] The sample consisted of 58 phishing scams inventoried by Hacker Factor between 14-Jan-2005 and 2-Apr-2005. Internet Explorer accounted for 15 (25%) of the inventoried scams. Other phishing archives have shown IE usage as high as 80%. The variability depends on multiple factors including the number of groups observed, number of unique scams received, and the total collection size. Five phishing groups were identified within the 58 samples. Two phishing groups that used 'wget' (or similar tool) generated more phishing emails and heavily biased this collection volume. The remaining three groups (60%) used IE.

- **Frames**. Web pages that use encoded HTML frames will not have frame definitions available. The frames cannot be easily decoded and converted back to HTML.

- **Images**. Copied web pages will not have the images available. Images must be copied separately, and hyperlinks may need to be corrected.

- **Extended HTML**. Although the preview window of these tools displays decoded HTML, copying content from the preview pane does not copy anything except the visible HTML.

- **JavaScript**. All JavaScript on the web page is lost. The same applies to Java and ActiveX components. For many e-commerce sites, JavaScript plays an important part in defining their look-and-feel; without JavaScript, users will notice.

Beyond the technical limitations, most commercial web editing tools are expensive and generally unavailable to non-commercial developers. Phishers usually do not have access to these tools.

2.4.4 Mirror Limitations

Nearly all web-mirroring tools have the same fundamental limitation: they cannot decode JavaScript. JavaScript is an actual programming language. In order to identify the output, the program must execute. Unfortunately, there is no way to determine beforehand if the program is semantically correct or will ever terminate – this is called the Turing problem. Because the JavaScript execution may hang, run indefinitely, or require external triggers (e.g., user input) in order to complete, nearly all web-mirroring tools make no attempt to process JavaScript – HTML is mirrored with the original raw/unedited JavaScript. The same limitation applies to other web-based applications including Java, ActiveX, and Macromedia's Flash[2].

[2] To our knowledge, Templeton is the only web-mirroring tool capable of identifying and mirroring static URLs in Flash files. Templeton does not process dynamic URLs in Flash files.

3 Solution Details: Page Encoding

Encoding a web page in JavaScript prevents trivial web page mirroring. The JavaScript code essentially draws the web page, while preventing interpretation by web mirroring tools. The process takes two phases:

- **Phase 1: Wrapping**. A simple JavaScript wrapper around a plain-text web page prevents mirroring and automatic copying of images and style sheets. But, the wrapper does not significantly deter phishers; they will simply remove the JavaScript wrapper and edit the HTML. Mirroring images and style sheets from the wrapped web pages must be done manually, creating a minor inconvenience for phishers.

- **Phase 2: Encoding**. A simple encoding, such as Unicode (converting characters to their hexadecimal equivalent) is sufficient to block most web mirroring tools and be complicated enough to deter less-technical phishers. In order to modify a web page, it must first be decoded.

Appendix B shows an example of this process, where the homepage for Google is provided in both plain (unencoded) HTML and JavaScript encoded. Any mirroring tool can readily copy the plain web page and associated images (logo.gif). However, simply encoding the web page into Unicode renders the page unmirrorable by automated tools, and too complex to trivially edit without decoding.

Both web pages (plain and encoded) render the same web page with the same images, properties, etc. The only difference is that the encoded page cannot be mirrored as easily. Attempting to mirror the web page using Internet Explorer or equally simple means will copy the encoded page, but not decode it and not mirror any associated images or style sheets. Similarly, viewing the HTML source code in the browser will only show the encoded page.

The encoding process may be performed in JavaScript, Java, ActiveX, Flash, or other encoded web system. Of these options, JavaScript is almost universally available for all browsers. The encoding process can be implemented in two main ways:

- **Static**. Each web page is encoded once and then uploaded to the web server. All browsers receive the same encoded web page.

- **Dynamic**. Each web page is encoded on-the-fly and in real-time by a CGI script or web server plug-in. Using this approach, different users (browsers) may receive different encodings of the same web page.

3.1 Benefits of Encoding

The primary benefit from encoded web pages is the inability to readily mirror the pages. This prevents unskilled phishers from copying and modifying a target company's public appearance. Other benefits include:

- **Speed**. The encoding function, even very complicated variations, operate in real-time and with little or no impact to the web server. If pages are statically encoded, then there is no speed impact at all.

- **Page size**. Although Unicode essentially triples the size of the web page (3 bytes for every character), other encoding systems, such as simple substitution ciphers, will effectively maintain the data size. For large web pages, the encoded content can be compressed and rely on a decompression function written in JavaScript.

- **Complexity**. The example in this document uses a simple Unicode encoding. Most programmers can readily reverse this and develop automated 'decode' tools. More complex systems, such as XOR, addition, substitution, and dynamic schemes can be easily applied. The more dynamic and complex the encoding, the more effort it will take to decode.

- **Defendable**. Encoding developers can create logic bombs to ensure standalone JavaScript interpreters will be unable to decode the data – essentially turning the decoding into a Turing test (NP-complete; or in

English, really hard to automate decoding). A single decoding system should be unable to generically reverse complex encoding systems.

- **Implementation options**. The encoding can be implemented as a stand-alone application, CGI, or plug-in to a web server. Current benchmarks show that a plug-in to Apache 1.*x* delivers no measurable performance degradation.[3] In contrast, a standalone CGI application creates latency during initialization. A static encoding system can be implemented independently of the web server.

- **Hosting**. While many companies use third-party hosting sites, such as Akamai, to cache images, these companies generally provide their own HTML pages. The proposed encoding system only applies to HTML code, so a dynamic encoding system will not impact third-party hosting sites. Alternately, web pages can be statically encoded and stored on third-party servers.

- **Decoding algorithm**. The decoding algorithm must be present in the JavaScript. This means that strong cryptography is just as effective as a weak-encoding system. (If the person can decode a weak encoding, then they can also decode strong encryption.) The only issue becomes the phisher's ability to automate the decoding process. Applying weak ciphers directly corresponds to faster JavaScript processing times, and simple encoding systems have no restrictions from exporting cryptographic munitions.

- **Adoption**. Page encoding does not require every phishing target to adopt it in order to be effective. If there are few adopters, then the solution will be more effective for the few companies as they become less desirable targets. Widespread adoption will drive a phishing-market need for solutions around this technique.

3.2 Limitations to Encoding

The encoding approach does have a few limitations.

- **JavaScript**. Users without JavaScript enabled will be unable to view JavaScript-encoded web pages. While many web sites do not currently require JavaScript, a few – but significant – web sites currently demand browsers with JavaScript enabled. JavaScript offers an interesting security tradeoff: it permits page encoding, but has historically been insecure under IE and Netscape.

  Not every web page needs to be encoded. Because only certain web pages are regularly mirrored, such as login pages, adoption will not necessarily result in the entire site becoming inaccessible. Only potential phishing victims – those with login credentials – will be required to use scripting. In many cases, sites require scripting for logins, so enforcing its use on the login page is minor and equivalent to requiring an updated browser and other defensive options.

- **Web indexing services**. Search engine services such as Google and AltaVista will be unable to index the web site. One possible workaround is to provide an encoded web page to normal browsers, but a plain web page to browsers identifying themselves as Googlebot or other indexing services. Although a skilled phisher could imitate a permitted indexing service by forging the HTTP "User-Agent" field, an unskilled individual would be deterred.

[3] Benchmarks were conducted on a 550MHz Celeron running RedHat 7.3 with Apache 1.2.29. A slow CPU was selected so minor processing changes would accumulate into significant time delays. Benchmarking was performed using httperf-0.8 with the command-line '`httperf --client=0/1 --server=localhost --port=80 --uri=/ --rate=10 --send-buffer=4096 --recv-buffer=16384 --num-conns=1000 --num-calls=1`'. The tests were conducted on the same system and not over a network; network delays were not a benchmarking factor. The experiment was conducted 10 times for both encoded and plain pages in order to determine a standard deviation. Processing 10,000 encoded page requests took no measurable time difference compared to plain page requests. Both tests reported 99.903 seconds (range from 99.902 to 99.905) with a standard deviation of 0.0011 seconds.

For most e-commerce sites, web indexing of anything beyond a few external web pages (front pages, services, FAQ, etc.) is unessential and undesirable. Page encoding offers an additional method to prevent indexing web robots.

- **Existing scams**. Page encoding has no impact on existing mirrors and phishing scams. Encoding only deters the creation of new mirrors and phishing scams.

- **Phishing malware**. This solution makes no effort to address spyware and phishing malware.

- **Static encoding algorithms**. If the encoding algorithm never changes, then automated systems can be created for mirroring the site. A programmer can develop an automated system in under 24 hours and we estimate mass-disseminate to take about a week, essentially giving an encoding a minimum lifespan of one week. The encoding system should change weekly or more frequently. Changes could be simple, such as new bit-wise combination patterns, or more complex, such as segmented and recursive decoding functions. An ideal system would use a suite of algorithmic systems and combine them randomly.

- **Testing**. Prior to deployment, nearly all companies test new web pages. Tests frequently use automated systems. Testing encoded web pages requires the ability to decode the web page. Fortunately, encoding systems are provable: regardless of the web content, the decoded output matches the input to the encoding function. If this style of proof is acceptable, then testing only needs to occur on original, un-encoded web pages. Manual testing prior to the initial deployment can resolve most concerns.

- **Page interception**. In theory, a web browser plug-in could be developed for capturing decoded web pages. JavaScript stores decoded pages and images in internal variables. The plug-in would intercept the JavaScript processor and access the current variable states. Due to the JavaScript sandbox[4], this type of access would constitute a breach in the security of the JavaScript environment. This type of program requires very specialized skills and would be difficult to create; we estimate a minimum of 3 months for development.

3.3 Anticipated Evolution

Phishers evolve around anti-phishing solutions. The rate of evolution depends on the companies that implement the solution, and the degree of variety in the implementations. Adoption by primary phishing targets (e.g., eBay, PayPal, Citibank) will drive the evolutionary process. Secondary targets (e.g., Regions, Washington Mutual, Suntrust, and Citizens) will impact the evolutionary rate, but not as dramatically as the primary targets. Tertiary targets (e.g., Wells Fargo, Comcast, and Cox) are likely to have little or no impact on the evolutionary cycle beyond driving phishers away from their respective sites.

The evolutionary steps, assuming mass adoption with significant variation, are expected to follow the three-phase life cycle described in Sections 3.3.1, 3.3.2, and 3.3.3. A lower adoption rate will slow the evolutionary process, and a lower variety will increase the evolutionary process. The single exception is eBay – this company that is very profitable for phishers due to its various uses for laundering as well as other scams. Adoption by eBay could single-handedly speed the evolutionary process.

3.3.1 Short term

From the point of deployment until about 6 months after deployment, page encoding will likely deter phishers from sites that adopt this solution.

[4] JavaScript runs in an enclosed environment – it should not be able to access anything outside of the environment, and nothing outside should access anything inside. For this reason, independent web windows using JavaScript cannot (normally) communicate. The notable exceptions include user input (requiring a user), applet communication (requiring the JavaScript to reach out to a Java or ActiveX application), and cookies (which do not permit access to the displayed web page and cannot normally be shared by different servers).

- **Migration away**. Initially, phishers will migrate toward more desirable (easier) targets. The initial migration will include moving away from more profitable targets.

- **Old code**. Along with migrating away, old mirrors will continue to be used. Existing pre-packaged phishing scams will likely be used much longer than the current usage durations. These scams will become increasingly out of date and less credible to potential victims.

3.3.2 Mid term

Between 3 and 12 months, a new market will open for skilled phishers.

- **Market swing**. The trading value for working scams against these target companies will increase. The demand for working scams will surpass the ability to fulfill the demand. The underground market will reach a bottleneck.

  - **Less desirable**. If only less desirable targets implement this solution, then they will not be targeted.

  - **More desirable**. If more desirable targets implement page encoding, then cost for these phishing servers will drive market value. Less experienced phishers will move to easier targets, while technically skilled phishers will create scams with higher trading values.

- **Fresh meat**. New/novice phishers will continue to target unprotected sites. If encoding becomes widespread, then inexperience phishers will likely leave for other online scams, such as advanced fee fraud (a.k.a. Nigerian scams or "419") or simple mass mailings (spam, spim, etc.) with fraudulent offers.

- **Alternate sources**. Alternate mirror sources, such as Archive.org and Google's cache, will be used for acquiring phishing templates.

- **Homogenous scams**. Because fewer people will have the skills needed to create new phishing scams, the current variety is likely to decrease. This opens the possibility for tracking and identifying the individuals responsible for creating the scams. This also lessens the effectiveness of new scams.

3.3.3 Long term

Beyond 9 months, workarounds and a new skill set will develop.

- **New niche**. Phishing groups are segmented into required skills. The skills include scam creation, server acquisition, mass mailing, validation, and laundering. A new niche will develop based around the ability to acquire and decode the encoded sites. This very narrow skill will become a bottleneck for phishing development. This skill will also become an expensive service – possibly more expensive than money laundering[5].

- **Plug-ins**. If there is wide adoption, plug-ins and other capture tools will appear for mirroring pages decoded by web browsers. Initially, we expect these to be held tightly by the developers, but eventually released (stolen, sold, etc.) and shared among the phishing community. These are unlikely to be widely available until at least 18 months from the initial page encoding deployment. The first revisions of these tools are likely to have complexity on par with manually decoding – beyond the skill set of most phishers. As with most malware evolutionary cycles, ease of use generally takes 6-12 months from the initial release.

  Although this type of workaround defeats page encoding, the estimated 24 month minimum deployment provides ample time to analyze the evolving threat. Additionally, page encoding will still deter new phishers who do not have access to a capture tool.

[5] Money laundering services generally collect between 10% and 40% of the cashed amounts.

4 About Hacker Factor

The company, Hacker Factor, was established in 2002. Our services cover all stages of computer security risk mitigation: assessment, evaluation, detection, solutions, and forensics. Beyond consulting and forensics, we are actively involved in research projects that aim to improve online security. Current projects at Hacker Factor include anti-spam and anti-phishing research, malware tracking, and forensics tool development. In addition, we develop general security-oriented solutions.

5 Appendix A: Sample Code

The following source code provides an example web page encoding system. This source code converts HTML into Unicode with a JavaScript decoder. More complicated encoding systems may use compression, encryption, or combinations of logic operators (and, or, xor, shift, etc.). While this example employs a one-pass encoding scheme, more complex systems could use multiple passes.

The HTML code generated by this sample program defeats web site mirroring tools such as most web browser's "Save As" options (Internet Explorer, Firefox, Opera, Safari, Mozilla, etc.), and command-line mirroring tools such as 'wget'. While these tools will copy the encoded web page, they will not mirror images nor decode the web page.

```
/*********************************************************
 HTML to Javascript
 Copyright 2005 Hacker Factor, all rights reserved.
 Created 28-March-2005.

 This software is provided as an open source, proof-of-concept.
 There is no warranty, expressed or implied.  Use at your own risk.

 This code is not freeware.  It is provided as donation-ware.
 If you find this code useful, either in a project or as a concept,
 please send a donation to: http://www.hackerfactor.com/contact.html

 This code may not be used for commercial purposes without written permission.

 Credit:
 I got the idea on 25-March-2005 after looking over an encoded spyware
 web page.  A few of the underground phishers/spyware/virus groups use
 encoded web pages to prevent mirroring and theft by competing groups.
 It occurred to me that this same use model would deter these same groups
 from mirroring legitimate web pages.

 This code is provided as a stand-alone application.
 To use it:
   Compile:  cc html2javascript.c -o html2javascript
   To run:   ./html2javascript < oldfile.html > newfile.html
 *********************************************************/
#include <stdlib.h>
#include <stdio.h>

int     main    ()
{
  int c;        /* character from stdin */

  /* Print the HTML header */
  printf("<HTML>\n");
  printf("<NOSCRIPT>This page requires JavaScript</NOSCRIPT>\n");
  printf("<SCRIPT LANGUAGE=\"JavaScript\"><!--\n");

  /* Print the JavaScript decoder */
  printf("document.write(unescape(\"");

  /* Encode the HTML -- very simple: unicode */
  /* More complex can use XOR, Addition, Bit shifting, etc. */
  while(!feof(stdin))
    {
    c=fgetc(stdin);
    if (c != -1) printf("%%%02X",c);
    }

  /* End JavaScript decoder */
  printf("\"));\n");
  printf("--></SCRIPT>\n");
  return(0);
} /* main() */
```

6 Appendix B: Example Encoding

Page encoding starts with a plain/unencoded web page (Table 1). The page is wrapped in JavaScript and encoded (Table 2). The web browser received the encoded web page and executes the JavaScript. The JavaScript decodes and renders the original web page.

Table 1. Sample unencoded web page from the URL <http://www.google.com/>.

```
<html><head><meta http-equiv="content-type" content="text/html;
 charset=UTF-8"><title>Google</title><style><!--
body,td,a,p,.h{font-family:arial,sans-serif;}
.h{font-size: 20px;}
.q{color:#0000cc;}
//-->
</style>
<script>
<!--
function sf(){document.f.q.focus();}
// -->
</script>
</head><body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b alink=#ff0000
onLoad=sf()><center><img src="/intl/en/images/logo.gif" width=276 height=110
alt="Google"><br><br>
<form action=/search name=f><script><!--
function qs(el) {if (window.RegExp && window.encodeURIComponent) {var
qe=encodeURIComponent(document.f.q.value);if (el.href.indexOf("q=")!=-1)
{el.href=el.href.replace(new RegExp("q=[^&$]*"),"q="+qe);} else {el.href+="&q="+qe;}}return 1;}
// -->
</script><table border=0 cellspacing=0 cellpadding=4><tr><td nowrap><font size=-
1><b>Web</b>    <a id=1a class=q href="/imghp?hl=en&tab=wi" onClick="return
qs(this);">Images</a>    <a id=2a class=q href="http://groups-
beta.google.com/grphp?hl=en&tab=wg" onClick="return
qs(this);">Groups</a>    <a id=4a class=q href="/nwshp?hl=en&tab=wn"
onClick="return qs(this);">News</a>    <a id=5a class=q
href="/frghp?hl=en&tab=wf" onClick="return qs(this);">Froogle</a>    <a id=7a
class=q href="/lochp?hl=en&tab=wl" onClick="return qs(this);">Local</a><sup><a
href="/lochp?hl=en&tab=wl" style="text-decoration:none;" onclick="return qs(this);"><font
color=red>New!</font></a></sup>    <b><a href="/options/index.html"
class=q>more &raquo;</a></b></font></td></tr></table><table cellspacing=0
cellpadding=0><tr><td width=25%> </td><td align=center><input type=hidden name=hl
value=en><input maxLength=256 size=55 name=q value=""><br><input type=submit value="Google
Search" name=btnG><input type=submit value="I'm Feeling Lucky" name=btnI></td><td valign=top
nowrap width=25%><font size=-2>  <a href=/advanced_search?hl=en>Advanced
Search</a><br>  <a href=/preferences?hl=en>Preferences</a><br>  <a
href=/language_tools?hl=en>Language Tools</a></font></td></tr></table></form><br><br><font size=-
1><a href="/ads/">Advertising Programs</a> - <a href=/services/>Business Solutions</a> - <a
href=/intl/en/about.html>About Google</a></font><p><font size=-2>&copy;2005 Google - Searching
8,058,044,651 web pages</font></p></center></body></html>
```

Table 2. Encoded web page of <http://www.google.com/>. The JavaScript decoder is highlighted in bold.

```
<HTML>
<NOSCRIPT>This page requires Javascript</NOSCRIPT>
<SCRIPT LANGUAGE="JavaScript"><!--
document.write(unescape("%3C%68%74%6D%6C%3E%3C%68%65%61%64%3E%3C%6D%65%74%61%20%
68%74%74%70%2D%65%71%75%69%76%3D%22%63%6F%6E%74%65%6E%74%2D%74%79%70%65%22%20%63
%6F%6E%74%65%6E%74%3D%22%74%65%78%74%2F%68%74%6D%6C%3B%20%63%68%61%72%73%65%74%3
D%49%53%4F%2D%38%38%35%39%2D%31%22%3E%3C%74%69%74%6C%65%3E%47%6F%6F%67%6C%65%3C%
2F%74%69%74%6C%65%3E%3C%73%74%79%6C%65%3E%3C%21%2D%2D%0A%62%6F%64%79%2C%74%64%2C
%61%2C%70%2C%2E%68%7B%66%6F%6E%74%2D%66%61%6D%69%6C%79%3A%61%72%69%61%6C%2C%73%6
1%6E%73%2D%73%65%72%69%66%3B%7D%0A%2E%68%7B%66%6F%6E%74%2D%73%69%7A%65%3A%20%32%
30%70%78%3B%7D%0A%2E%71%7B%63%6F%6C%6F%72%3A%23%30%30%30%30%63%63%3B%7D%0A%2F%2F
%2D%2D%3E%0A%3C%2F%73%74%79%6C%65%3E%0A%3C%73%63%72%69%70%74%3E%0A%3C%21%2D%2D%0
A%66%75%6E%63%74%69%6F%6E%20%73%66%28%29%7B%64%6F%63%75%6D%65%6E%74%2E%66%2E%71%
2E%66%6F%63%75%73%28%29%3B%7D%0A%2F%2F%20%2D%2D%3E%0A%3C%2F%73%63%72%69%70%74%3E
%0A%3C%2F%68%65%61%64%3E%3C%62%6F%64%79%20%62%67%63%6F%6C%6F%72%3D%23%66%66%66%6
6%66%66%20%74%65%78%74%3D%23%30%30%30%30%30%30%20%6C%69%6E%6B%3D%23%30%30%30%30%
63%63%20%76%6C%69%6E%6B%3D%23%35%35%31%61%38%62%20%61%6C%69%6E%6B%3D%23%66%66%30
```

```
%30%30%30%20%6F%6E%4C%6F%61%64%3D%73%66%28%29%3E%3C%63%65%6E%74%65%72%3E%3C%69%6
D%67%20%73%72%63%3D%22%2F%69%6E%74%6C%2F%65%6E%2F%69%6D%61%67%65%73%2F%6C%6F%67%
6F%2E%67%69%66%22%20%77%69%64%74%68%3D%32%37%36%20%68%65%69%67%68%74%3D%31%31%30
%20%61%6C%74%3D%22%47%6F%6F%67%6C%65%22%3E%3C%62%72%3E%3C%62%72%3E%0A%3C%66%6F%7
2%6D%20%61%63%74%69%6F%6E%3D%2F%73%65%61%72%63%68%20%6E%61%6D%65%3D%66%3E%3C%74%
61%62%6C%65%20%62%6F%72%64%65%72%3D%30%20%63%65%6C%6C%73%70%61%63%69%6E%67%3D%30
%20%63%65%6C%6C%70%61%64%64%69%6E%67%3D%34%3E%3C%74%72%3E%3C%74%64%20%6E%6F%77%7
2%61%70%3E%3C%66%6F%6E%74%20%73%69%7A%65%3D%2D%31%3E%3C%62%3E%57%65%62%3C%2F%62%
3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%61
%20%69%64%3D%31%61%20%63%6C%61%73%73%3D%71%20%68%72%65%66%3D%22%2F%69%6D%67%68%7
0%3F%68%6C%3D%65%6E%26%74%61%62%3D%77%69%26%69%65%3D%55%54%46%2D%38%22%3E%49%6D%
61%67%65%73%3C%2F%61%3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26
%6E%62%73%70%3B%3C%61%20%69%64%3D%32%61%20%63%6C%61%73%73%3D%71%20%68%72%65%66%3
D%22%68%74%74%70%3A%2F%2F%67%72%6F%75%70%73%2D%62%65%74%61%2E%67%6F%6F%67%6C%65%
2E%63%6F%6D%2F%67%72%70%68%70%3F%68%6C%3D%65%6E%26%74%61%62%3D%77%67%26%69%65%3D
%55%54%46%2D%38%22%3E%47%72%6F%75%70%73%3C%2F%61%3E%26%6E%62%73%70%3B%26%6E%62%7
3%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%61%20%69%64%3D%34%61%20%63%6C%61%
73%73%3D%71%20%68%72%65%66%3D%22%2F%6E%77%73%68%70%3F%68%6C%3D%65%6E%26%74%61%62
%3D%77%6E%26%69%65%3D%55%54%46%2D%38%22%3E%4E%65%77%73%3C%2F%61%3E%26%6E%62%73%7
0%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%61%20%69%64%3D%35%
61%20%63%6C%61%73%73%3D%71%20%68%72%65%66%3D%22%2F%66%72%67%68%70%3F%68%6C%3D%65
%6E%26%74%61%62%3D%77%66%26%69%65%3D%55%54%46%2D%38%22%3E%46%72%6F%6F%67%6C%65%3
C%2F%61%3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26%6E%62%73%70%
3B%3C%61%20%69%64%3D%37%61%20%63%6C%61%73%73%3D%71%20%68%72%65%66%3D%22%2F%6C%6F
%63%68%70%3F%68%6C%3D%65%6E%26%74%61%62%3D%77%6C%26%69%65%3D%55%54%46%2D%38%22%3
E%4C%6F%63%61%6C%3C%2F%61%3E%3C%73%75%70%3E%3C%61%20%68%72%65%66%3D%22%2F%6C%6F%
63%68%70%3F%68%6C%3D%65%6E%26%74%61%62%3D%77%6C%26%69%65%3D%55%54%46%2D%38%22%20
%73%74%79%6C%65%3D%22%74%65%78%74%2D%64%65%63%6F%72%61%74%69%6F%6E%3A%6E%6F%6E%6
5%3B%22%3E%3C%66%6F%6E%74%20%63%6F%6C%6F%72%3D%72%65%64%3E%4E%65%77%21%3C%2F%66%
6F%6E%74%3E%3C%2F%61%3E%3C%2F%73%75%70%3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%26
%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%62%3E%3C%61%20%68%72%65%66%3D%22%2F%6F%70%7
4%69%6F%6E%73%2F%69%6E%64%65%78%2E%68%74%6D%6C%22%20%63%6C%61%73%73%3D%71%3E%6D%
6F%72%65%26%6E%62%73%70%3B%26%72%61%71%75%6F%3B%3C%2F%61%3E%3C%2F%62%3E%3C%2F%66
%6F%6E%74%3E%3C%2F%74%64%3E%3C%2F%74%72%3E%3C%2F%74%61%62%6C%65%3E%3C%74%61%62%6
C%65%20%63%65%6C%6C%73%70%61%63%69%6E%67%3D%30%20%63%65%6C%6C%70%61%64%64%69%6E%
67%3D%30%3E%3C%74%72%3E%3C%74%64%20%77%69%64%74%68%3D%32%35%25%3E%26%6E%62%73%70
%3B%3C%2F%74%64%3E%3C%74%64%20%61%6C%69%67%6E%3D%63%65%6E%74%65%72%3E%3C%69%6E%7
0%75%74%20%74%79%70%65%3D%68%69%64%64%65%6E%20%6E%61%6D%65%3D%68%6C%20%76%61%6C%
75%65%3D%65%6E%3E%3C%69%6E%70%75%74%20%74%79%70%65%3D%68%69%64%64%65%6E%20%6E%61
%6D%65%3D%69%65%20%76%61%6C%75%65%3D%22%49%53%4F%2D%38%38%35%39%2D%31%22%3E%3C%6
9%6E%70%75%74%20%6D%61%78%4C%65%6E%67%74%68%3D%32%35%36%20%73%69%7A%65%3D%35%35%
20%6E%61%6D%65%3D%71%20%76%61%6C%75%65%3D%22%22%3E%3C%62%72%3E%3C%69%6E%70%75%74
%20%74%79%70%65%3D%73%75%62%6D%69%74%20%76%61%6C%75%65%3D%22%47%6F%6F%67%6C%65%2
0%53%65%61%72%63%68%22%20%6E%61%6D%65%3D%62%74%6E%47%3E%3C%69%6E%70%75%74%20%74%
79%70%65%3D%73%75%62%6D%69%74%20%76%61%6C%75%65%3D%22%49%27%6D%20%46%65%65%6C%69
%6E%67%20%4C%75%63%6B%79%22%20%6E%61%6D%65%3D%62%74%6E%49%3E%3C%2F%74%64%3E%3C%7
4%64%20%76%61%6C%69%67%6E%3D%74%6F%70%20%6E%6F%77%72%61%70%20%77%69%64%74%68%3D%
32%35%25%3E%3C%66%6F%6E%74%20%73%69%7A%65%3D%2D%32%3E%26%6E%62%73%70%3B%26%6E%62
%73%70%3B%3C%61%20%68%72%65%66%3D%2F%61%64%76%61%6E%63%65%64%5F%73%65%61%72%63%6
8%3F%68%6C%3D%65%6E%3E%41%64%76%61%6E%63%65%64%20%53%65%61%72%63%68%3C%2F%61%3E%
3C%62%72%3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%61%20%68%72%65%66%3D%2F%70%72
%65%66%65%72%65%6E%63%65%73%3F%68%6C%3D%65%6E%3E%50%72%65%66%65%72%65%6E%63%65%7
3%3C%2F%61%3E%3C%62%72%3E%26%6E%62%73%70%3B%26%6E%62%73%70%3B%3C%61%20%68%72%65%
66%3D%2F%6C%61%6E%67%75%61%67%65%5F%74%6F%6F%6C%73%3F%68%6C%3D%65%6E%3E%4C%61%6E
%67%75%61%67%65%20%54%6F%6F%6C%73%3C%2F%61%3E%3C%2F%66%6F%6E%74%3E%3C%2F%74%64%3
E%3C%2F%74%72%3E%3C%2F%74%61%62%6C%65%3E%3C%2F%66%6F%72%6D%3E%3C%62%72%3E%3C%62%
72%3E%3C%66%6F%6E%74%20%73%69%7A%65%3D%2D%31%3E%3C%61%20%68%72%65%66%3D%22%2F%61
%64%73%2F%22%3E%41%64%76%65%72%74%69%73%69%6E%67%26%6E%62%73%70%3B%50%72%6F%67%7
2%61%6D%73%3C%2F%61%3E%20%2D%20%3C%61%20%68%72%65%66%3D%2F%69%6E%74%6C%2F%65%6E%
2F%61%62%6F%75%74%2E%68%74%6D%6C%3E%41%62%6F%75%74%20%47%6F%6F%67%6C%65%3C%2F%61
%3E%3C%2F%66%6F%6E%74%3E%3C%70%3E%3C%66%6F%6E%74%20%73%69%7A%65%3D%2D%32%3E%26%6
3%6F%70%79%3B%32%30%30%35%20%47%6F%6F%67%6C%65%20%2D%20%53%65%61%72%63%68%69%6E%
67%20%38%2C%30%35%38%2C%30%34%34%2C%36%35%31%20%77%65%62%20%70%61%67%65%73%3C%2F
%66%6F%6E%74%3E%3C%2F%70%3E%3C%2F%63%65%6E%74%65%72%3E%3C%2F%62%6F%64%79%3E%3C%2
F%68%74%6D%6C%3E"));
</SCRIPT>
```