2 The Characteristics of Spam Email

It is easy for a person to look at a piece of email and say, "This isn't something I asked for. It looks like an advertisement, and I don't want it, so it must be spam." But although it is easy for humans to recognize spam, it is much harder for software to recognize it. And, after all, the point of spam-blocking software is to eliminate the need for humans to recognize spam.

In this chapter we discuss spam as it appears to software and not as it appears to a human. For example, Figure 2.1 shows how a piece of spam email might appear to you when viewed in your email reader.

The message in Figure 2.1, however, might look like the following to software:

```
<a href="dvds.example.com"><img src="dvds.example.com/images/em1"></a>
```

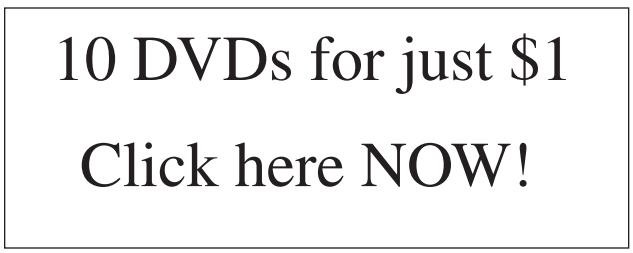

FIGURE 2.1 How spam email might appear to a human.

This is a form of HTML code called a **clickable link**, and although software lacks human eyes, it can easily detect that this is a clickable link.

In this chapter we reveal, at the code level, how spam email is often internally structured. But bear in mind that spam email is *constantly changing and evolving*, and the lessons you learn in this chapter—although forward-looking—are not the total picture. Thus, we encourage you to collect spam to examine for yourself. By doing so, you will note trends not covered here.

One way to do this is to collect spam in your normal mailbox, but because this can get a bit messy, we instead advise you to set up a bait machine (see Chapter 3).[1]

2.1 CONNECTION BEHAVIOR

Spam email is sent like all other email. The sending site connects to the local sendmail daemon and waits for the local sendmail to say it is ready.

Sender connects and waits for:
```
220 local.hostname ...
```

Here, the line beginning with `220` is from the local sendmail and tells the sending client that it can begin to send messages.

A well-behaved client will wait for the `220` before sending anything, but some spamming software will not. To speed up the spamming process, some spamming software will send the entire message without waiting for the `220`.

Sender connects and immediately sends the whole message.
Then it disconnects and moves on to the next host in line.
```
220 local.hostname...
```

This works for spammers because email runs on top of a protocol called Transmission Control Protocol/Internet Protocol (TCP/IP). With TCP/IP the operating system sets up a buffer to hold the incoming message, and because sendmail looks only at the buffer, rather than at the actual connection, it does not see that the inbound sender has already sent everything. Thus, sendmail will obligingly accept the message and probably deliver it.

1. Or you may prefer to simply collect spam in your own mailbox.

Fortunately, version 8.13 of sendmail adds a new feature to catch just this sort of ill-mannered behavior. Called the `greetpause` `FEATURE`, it causes sendmail to sleep briefly before sending the `220`. After the sleep and before sending the `220`, sendmail checks to see whether input from the sender is already present in the buffer. If it is, sendmail thereafter rejects all SMTP commands from that sender, for that connection, issuing a `554` error.

Naturally it would seem better to just drop the connection, but sendmail dares not do that because doing so could leave unread information in the TCP buffer, eventually filling that buffer.

In the future, spamming sites will probably adapt to this form of detection, modify their behavior, and start sending only after receiving the `220` greeting, after the `EHLO`, after the `MAIL FROM`, or after a `RCPT TO`. To adapt, spamming software will need to move its asynchronous attack further into the synchronous protocol. As this competition continues, spamming software will slowly be forced to evolve into normal, well-behaved email sending software.

2.2 RELAYING THROUGH MX SERVERS

It is important to realize that mail is not sent directly to the host whose name follows the `@` in the email address. Instead, that host name is first looked up using DNS to find the names of its MX (mail exchange) servers. To illustrate, consider the email address `bob@example.com`. Before mail is sent to `example.com`, that host's MX records are looked up:

```
example.com.    MX 10 mail.example.com.
example.com.    MX 20 bob.fallback.host.
```

Here, the host `example.com` has two MX records, each beginning with the literal MX. Each MX is then followed by a number. The lower the number, the greater the preference given to deliver to that particular record. Following each number is the name of the host to which mail should be sent. Here, mail should be sent to `mail.example.com` first, and if that fails, it should be sent to `bob.fallback.host` instead.

MX records exist as a safety net so that mail will always be delivered somewhere. The idea in our example is that mail will be held by `bob.fallback.host` in the event that `mail.example.com` is down. Later, when `mail.example.com` returns to service, `bob.fallback.host` will forward the delayed mail to it.[2]

2. Actually, it is perfectly legal to have only one MX server (but it is discouraged to have none). However, we use two and more here for the sake of example.

Because normal email is delivered using the order specified by such MX records, mail from fallback hosts should be rare. But many spam sites take advantage of this behavior to avoid being rejected based on their connecting IP numbers. Their assumption is that spam email screening normally done on a main mail machine will not necessarily be done on a fallback machine, and—because most fallback services are provided by ISPs and so do not share the same policy or software of the main mail machine—this assumption is frequently correct.

Thus, many spam sites look up MX records and purposely send their spam email to the host with the highest number (lowest preference). When they do this, the main mail machine sees only the IP number of the fallback host, and the IP number of the spam sender remains hidden.[3]

If you intend to filter email based on connecting site IP numbers, you will need to arrange your MX records to specifically trap this sort of IP number subterfuge. Consider again our example of the site `example.com`, which has MX records that look like this:

```
example.com.   MX 10 mail.example.com.
example.com.   MX 20 bob.fallback.host.
```

If `example.com` is set up like that, much spam email will be sent to `bob.fallback.host` as spammers attempt to avoid IP-number suppression. Instead, `example.com` might be set up to detect spam sending IP numbers, like this:

```
example.com.   MX 10 mail.example.com.
example.com.   MX 20 bob.fallback.host.
example.com.   MX 30 mail.example.com.
example.com.   MX 40 mail.example.com.
```

Now, normal mail will be delivered just as before, and if `mail.example.com` is down or busy, mail will instead be delivered to `bob.fallback.host`, also just as before. If both are down or busy, mail will be deferred at sending sites. But now, spam sites that send to the highest MX record will instead send directly to `mail.example.com`, where IP detection and suppression is running. This minimizes the number of spam sites that will connect to `bob.fallback.host`.

3. It is actually available in `Received:` headers, but those headers are difficult to parse and cannot always be trusted.

Note that, to accomplish this, we use two high number records, both pointing to `mail.example.com`. Clearly, a savvy spammer would recognize this trick and adapt to it. A better way to organize MX records requires you to control your own domain records. Consider another way to set up the preceding records:

```
example.com.    MX 10 mail.example.com.
example.com.    MX 20 mx1.example.com.
example.com.    MX 30 mx2.example.com.
example.com.    MX 40 mx3.example.com.
```

Here, all MX hosts are under your control, so the spam sender must look up IP numbers for these records in order to figure out where each is actually pointing. In this case, the IP number for `mx1.example.com` would be the same as that for `bob.fallback.com`, and the IP numbers for `mx2.example.com` and `mx3.example.com` would both be the same as the IP number for `mail.example.com`.

Modern operating systems allow each network interface to be assigned multiple IP numbers. If yours is one such operating system, consider assigning a different IP number to each of `mx2.example.com` and `mx3.example.com`, where those two differ from the IP number for `mail.example.com`.

The more work you impose on spam-sending sites, the less effective they become and the more costly their operation will be. Because spam email is a low-margin business, even an incremental increase in their costs may drive some out of business.

2.3 FALSIFYING THE ENVELOPE SENDER ADDRESS

All Internet email is sent using a protocol called SMTP (Simple Mail Transfer Protocol). In SMTP, mail is sent in units called **envelopes**. First the address of the envelope sender is sent, followed by one or more envelope recipient addresses. Finally the actual message is sent, headers and body together. Note that the final recipient sees only the headers and body. Envelope information is generally visible only to sendmail and its Milters.

When a user replies to an email message, the reply is generally sent to the header sender—the address listed in the `From:` header. But when a message is bounced, the bounce reply is generally sent to the envelope sender address.

Spam sites have learned about this property of email and can use it to their advantage. Spammers know that a large percentage of the email they send will bounce, but they do not care about bounces. Even if an address in a list continues to bounce, spammers will not remove it, because the cost of cleaning their lists is too high.

Some spam detection software looks at the envelope sender on the theory that the envelope sender will point to the spam-sending site, because that is where bounced email goes. But to avoid this type of detection (and also the potentially high volume of bounce return messages), spam-sending sites often use a false envelope sender. In other words, they lie about their identity. And not only do they lie, they also forge. That is, because many MTAs (sendmail included) reject the envelope sender if the host part of the address does not exist, spammers often use real envelope sender addresses (but not their own).

The result is that some poor user somewhere in the world will receive all of a spam site's bounces. This is certainly evil, but it is not illegal. There is nothing in the SMTP standard that can cause a sending site to tell the truth in the envelope sender specification.

Because spamming sites never clean their address lists and because spamming sites often lie about the envelope sender, you should probably never bounce (reject) spam email. If you do, you may be unwittingly adding to the problem by bouncing to an innocent user. In general, it is better to accept and discard spam email or to accept and archive it.[4]

An alternative (if you have some way to keep count) is to bounce spam the first time it is received from a site. In that way, if good email is bounced, the sender (a real person) can contact someone at your site to have the problem corrected.[5]

2.4 DISGUISING THE `Subject:` HEADER

It is tempting to examine the `Subject:` header to identify spam email. Unfortunately, the `Subject:` header is entirely under the spammer's control and thus can never be trusted.

But not all spam senders are created equal. Some will do everything possible to make the `Subject:` header appear benign, but others are inept and will provide clues in that header.

Clearly, it can benefit you to try to pick off these inept spammers. But first, understand that, by doing so, you run the risk of identifying good email as spam. To illustrate, consider the following examples of actual spam `Subject:` headers:

```
Subject: Learn how thousands are making a fortune with eBay...
Subject: Earn While You Learn  sjdv
Subject: Visit PlayboyPlus for New Playmate Pics
```

4. We fudge for simplicity here. Actually a bounce occurs only when failure notification is *mailed* back to the envelope sender. A rejection does not necessarily generate a bounce.
5. In your bounce (reject) message, always try to include helpful information that can aid in correcting a wrongful rejection of good email.

```
Subject: Discreet delivery
Subject: naked and cute. watch my movie.
Subject: ,^refina'nce now and ;save-    r
Subject: Extend your auto warranty, free quote
Subject:
Subject: Spring Special
Subject: <a href="http://www.example.com/mp/axis/">or not
Subject: get the edge
Subject: Re:  jock
Subject: V1AGR*A final1y fOund a tOugh cOmpet1tOr -- CIAL1-S
Subject: Email Verification!Please take a look.  wyl
Subject: Celebrity Secret to looking young!
Subject: You've seen ads for Levitra on TV, Does it Work?i s k
Subject: Refinance your Home and skip a payment
Subject: Important notify about your e-mail account.
```

Although all of these appear to be spam `Subject:` headers, in actuality the two shown in bold were real email messages to users who wanted to receive them.

In general, it is unwise to employ a spam-screening strategy that examines only `Subject:` headers. Such screening is prone to errors, and as spammers mature, the use of such headers will likely decline. We expect spam email `Subject:` headers of the future to look more like these:

```
Subject: Yesterday was fun
Subject: Re: updating my address
Subject: Thanks again!
Subject: Email Statement
Subject: next appointment
Subject: I hope so
```

Another trick used by spam senders is to base64-encode the `Subject:` header. This has the advantage of making it hard for detection software to see the header, while still allowing the end user to see a readable subject line. Such base64 code might look like this:

```
Subject: =?iso-8859-1?b?OTAgZGF5cyB0byB5b3V0aA==?=
```

The end user will see this:

```
Subject: 90 days to youth
```

Unfortunately, however, base64 encoding of a `Subject:` does not always indicate a spam message, because it is also the only method that allows some foreign languages to insert legal headers into email. For example, in the United States a user might consider `=?.GB2312?` (which indicates simple Chinese) to be a solid indicator of spam email. But in China, that same encoding might indicate good email, whereas `=?US-ASCII?` might indicate spam. Thus, again, we recommend that you screen `Subject:` headers sparingly, if at all.

2.5 CAMOUFLAGING THE HTML BODY

HTML (Hypertext Markup Language) is a language that describes how text and objects should be displayed. HTML also controls the interaction between the user and the message. Most, but not all, email is formatted using HTML. This is a boon to users and businesses, which benefit from the ability to display attractive fonts and images as part of an email message. But it is also a boon to spam email senders because HTML-capable mail programs are liberal in what they accept, and that allows spam email to be easily disguised.

In this section we first describe the use of HTML comments as a means to disguise content. Then we describe two forms of encoding often used to disguise content: character-entity encoding and URL encoding.

2.5.1 HTML COMMENTS

Consider this HTML comment:

```
<!-- this is a comment -->
```

An HTML comment begins with the four-character sequence `<!--` and ends with the three-character sequence `-->`. Comments may span multiple lines and may enclose and hide arbitrary text.

Early in spam history, spammers discovered that words could be broken up with HTML comments, making those words appear normally when displayed. Here's an example:

```
V<!--THYME-->ia<!--ONION-->gr<!--ALMOND-->a
```

Here, the word "Viagra" is hidden among the names of various foods. But because the comments are ignored by HTML-capable mail programs, this text is displayed like this:

```
Viagra
```

More recently, spammers have discovered that any unknown HTML keyword in angle brackets acts just like a comment.[6] Thus the comment technique has been modified by spammers and now looks like this:

```
V<xyxxx>ia<ftmmwe>gr</gvwquu>a
```

Clearly, to eliminate comments from HTML you must now possess a full understanding of valid HTML keywords. For example:

`<font>`	*Yes, a keyword*
`<fonts>`	*No, not a keyword, but instead a comment*

Similarly, spam-screening software needs to know that `<p>`, `<b>`, and `<i>`, for example, are HTML keywords, but `<q>`, `<t>`, and `<c>` are not.

To make things more complicated, any words that begin with an `!` or a `?` and any non-HTML keywords that begin with a `/` character are also treated as comments:

`<!xxx>`	*Also a comment*
`<?xxx>`	*Also a comment*
`</xxx>`	*Also a comment*
`</font>`	*A keyword, not a comment*

Be aware, too, that intervening newlines may also be included in comments. Consider, for example, the following:

```
V<<OICE
GIMM>>I
A<<LIVE
```

6. We warned you that HTML-capable mail programs were far too forgiving.

```
ABLO>>G
R<<IVER
ITUN>>A
```

Here, we show the effect of both unbalanced angle brackets and a newline between the matching angle brackets of the comment. When viewed by HTML-capable mail programs, the preceding lines will look like this:

```
V<>I A<>G R<>A
```

In short, spammers can play many tricks to disguise HTML. Newlines may be included in comments. Extra angle brackets may become part of revealed messages.

As an aside, one trick that may be useful when you attempt to match angle brackets is to scan from the left to the right and look only ahead, never back. To illustrate, examine the following HTML fragment, in which we have indicated matching angle brackets using bold font:

```
<xxx<yyy<zzz<>www>
<aaa<bbb>ccc>ddd>eee>
```

Reading from left to right, note that the first left angle bracket starts the `xxx` comment. The following right angle bracket (even if there are intervening left angle brackets) ends the comment, revealing `www` as clear text. The next left angle bracket (that is not part of the comment) begins the `aaa` comment and ends just before `ddd`. The result, when viewed on HTML-capable mail programs, will look like this:

```
zzz> ccc>ddd>eee>
```

2.5.2 CHARACTER-ENTITY ENCODING

Character-entity encoding provides the means to include special characters in HTML documents. It can be invoked in either of two forms. Both begin with an ampersand character and end with a semicolon character. In between are either a keyword, such as `lt`, or a literal `#` followed by a three-digit decimal value. For example:

```
&lt;          A Keyword that produces a < character.
&#060;        # and three digits produces a < character.
```

Without character-entity encoding, the following special characters (to list only two) would not be possible:

`&copy;` *Produces the © symbol.*
`&#159;` *Produces the Ù symbol.*

The HTML standard allows all characters, and not just special characters, to be encoded in this way and provides a handy way for spammers to hide otherwise obvious information. Consider this example:

```
<a
href="mailto:&#098;&#111;&#098;&#064;&#101;&#120;&#097;&#109;
&#112;&#108;&#101;&#046;&#099;&#111;&#109;">
```

After decoding, the obscured email address is revealed to be an easily recognized address:

```
<a href="mailto:bob@example.com">
```

Clearly, it is necessary to decode character entities before attempting to parse HTML for clues about spam. In Section 11.4 we show you the C language code you can use to decode character-entity-encoded text.

2.5.3 URL ENCODING

A URL (Uniform Resource Locator) is a web reference. Every time you click on a link with your web browser, you are invoking a URL. The URL standard assigns special meanings to special characters. For example, a `?` character is used to separate a web reference from its arguments:

```
www.example.com/cgi-bin/search?name=bob
```

To prevent confusion, such special characters, when not in their special role, must be URL-encoded. To URL-encode a character, first prefix it with a `%` character, and then con-

vert the character itself into its two-digit hexadecimal ASCII value.[7] To illustrate, consider a search program whose name includes a `?` character:

```
big?search.pl
```

To use this program name as part of a URL, the `?` character must be converted into its hexadecimal value, `3F`, which must be preceded by a `%` character:

```
www.example.com/cgi-bin/big%3Fsearch.pl?name=bob
```

But as we have shown, almost anything will be misused by spammers to hide message content. Consider, for example, the following obscured web reference:

```
href="%77%77%77%2E%65%78%61%6D%70%6C%65%2E%63%6F%6D"
```

Clearly, before you can screen a message for spam content, you may need to URL-decode all such URL-encoded expressions.[8] The example, when decoded, will look like the following:

```
href="www.example.com"
```

In section 11.5, we show you the C language code you can use to decode URL-encoded text.

2.5.4 THE ORDER OF ENCODING

Character-entity decoding is performed by a web browser before it displays the text on the screen, but URL decoding is performed later, as a result of an HTML action, such as clicking on a web reference. Thus, web references may be first URL-encoded and then character-entity-encoded, and the result will be understood by HTML-capable mail programs. When decoding email to examine it for spam content, always character-entity-decode first and then URL-decode.

7. On UNIX machines, conversion from ASCII to hexadecimal is usually documented in the file `/usr/pub/ascii`. But it might alternatively be found in `/usr/share/lib/pub/ascii` or `/usr/share/misc/ascii`. Under Red Hat and Fedora Linux, it is found in the online manual `ascii(7)`.
8. When International Domain Names (IDNs) come into full use, decoding may no longer remain optional.

2.6 ATTEMPTING TO FOOL SIGNATURE DETECTORS

Signature detectors work by computing a value for a block (or chunk) of text in a message. For example, suppose a message contained this text:

```
We sell herbs at the lowest price you will
ever find on the net.
```

A signature-generating program might create an expression that represents this text, such as the following:

```
244372015810742154622705
```

This "signature" is saved in a file or database. Later, when another message arrives, its chunks are also given signatures. Then each signature is looked up, and, if it is found, that serves as an indication that the message may have been seen before. Clearly, several signatures will have to match so that one message may be considered significantly like another.

Similar spam detection software recognizes phonemes (distinct parts of words) instead of chunks of words. Other software performs permutations of the divided text to increase the number of signatures used, and still others perform statistical analysis on individual words and then store the probabilities.

But all these forms of spam detection share the common method of examining the message's text. Spammers, recognizing that text analysis is being used, have responded by adding large chunks of random text to each message.

Random text can be actual words and names in random order:

```
dissonant deanna heron aphasia restaurateur circulate
controllable corporeal cranston giuliano helmholtz bertha
albany shank eye asphyxiate commentary gaston aide filler
chipboard prostheses perturb cryptographer atlantic bernice
```

Random text can also be random combinations of characters:

```
eyhydxre yaceyaxv gesmveu vmlpv wmgrxa drgcah mqbjneq
wbfqzkmwr fdbkqogtgzwv lsunhut wuwnp- hivrkef dhdpfhcu
ndowgkx cjxrofun yepjhxp rhbxag ncgvmv
```

Random text can also be a solid stream of random characters:

```
hyfaqjimgdalmrymmolaktivajvctikdhpfzaplgumufsvtjgu
tccqenngjwtodktenkrvefpmkiherqymsccysqfbmapkkvxuo
tauimesuijmivglyefqlgclxvyjsxfgsfadrhvnrhzacfncmssx
awlzrjilipsbuuenbbdtievlmkpycivegidatnlccffyajnbmqw
```

Finally, random text can also be an actual abstract from real text, where a different abstract is used in each message:[9]

```
Mother called me home that night with a shout
that told me there was trouble. "Mom," I yelled,
pounding up the back steps. "What's wrong, Mom?"
```

When you parse a message to detect spam, your goal is to find a way to skip such random text and to run signatures only on the portions of the message that do not change. Portions of a message that should be checked include the following:

- Common images
- Web references
- Email addresses
- Phone numbers

2.7 UNNECESSARY ENCODING

MIME (Multipurpose Internet Mail Extensions), a protocol that defines how various objects may be embedded in email, is the standard used to create attachments. MIME also provides the standards for encoding attachments for mail transport.

When an attachment is binary (such as a photograph or a word processor document), it must be base64-encoded before email transport. When lines are overly long (as is much HTML code), that text must be quoted-printable-encoded before email transmission. Unfortunately, spammers have adopted these two valuable forms of encoding to help disguise spam email content.

9. As if spammers would care, random insertion of actual text abstracts runs the risk of copyright infringement.

The kind of MIME encoding used in an email message (if any) is specified by the `Content-Transfer-Encoding:` header:

```
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: quoted-printable
```

We will show how to decode base64 in section 11.2, and how to decode quoted-printable in section 11.3. Here we illustrate how they are used to disguise spam email. Consider this example:

```
Content-Type: text/html
Content-Transfer-Encoding: base64

PEZPTlQgZmFjZT0iVmVyZGFuYSIgc2l6ZT0zLjU+DQpDaW5kZXJlbGxhISAgUGV0ZXIgUGFuISAg
dWxsIG9mIGl0OyBlYXQgaXQgZXZlcnkgZGF5IGZvciBicmVha2Zhc3Q=
```

Here, the message content is of type HTML (the `text/html`), but rather than allow the HTML to be transmitted as is, the spammer has base64-encoded it to make it difficult to recognize. Clearly, you must first decode this base64-encoded text before you can screen it for spam.

In addition to its primary use (encoding long lines for transport), quoted-printable can be used to obscure text so that it is difficult to parse:

```
Content-Type: text/html
Content-Transfer-Encoding: quoted-printable

<br>
<a href=3D"http://bob.example.com/py/">
<img src=3D"http://amy.example.com/2/"></a>
```

Here, the internal equal signs have been turned into `=3D` expressions using quoted-printable encoding. You need to decode them back into equal signs before screening the message for spam content.

2.8 GROKKING THE SITE

Spam messages generally contain a method for the recipient to contact the spammer (or the business on whose behalf the spam is being sent). Some offer an email address and others a phone number, but most try to get the recipient to go to a website. To do this, spammers include a clickable web reference (URL) in the body of the spam email. But because spammers seek to keep their identities a secret, they generally try to disguise all web references. In this section we show some of the tricks they use.

First note that a web reference is usually specified as part of an HTML a command:

```
<a href="http://www.example.com"> visible text </a>
```

The `<a` (followed by a space) begins the command. The `<a` is followed by one or more keywords particular to that command, all terminated by the `>` character. The `href=` keyword indicates a web reference (URL). Following the `>` is the actual text that will appear on the screen and that must be clicked to invoke the web reference. The `</a>` ends the command.

In the following, we examine the pieces of this web reference one item at a time.

- Section 2.8.1 examines the leading `<a` part.
- Section 2.8.2 explains case insensitivity.
- Section 2.8.3 examines the `http:` part.
- Section 2.8.4 shows how email addresses can mask the `www.example.com` part.
- Section 2.8.5 shows that IP numbers and hexadecimal representations of IP numbers are used by spammers to disguise host names.
- Section 2.8.6 shows how the `www.example.com` part can be hidden with redirects.
- Section 2.8.7 shows how the `www.example.com` part can be ridiculously stretched out to say `aa.bb.cc.dd.ee.ff.gg.hh.ii.jj.kk.example.com`.
- Section 2.8.8 shows how the `www.example.com` part can be hidden behind CNAME records.
- Section 2.8.9 shows that URLs can also be used as comments.

One common method of using URLs to fight spam is to record the host names from those URLs in a database. Each time a new piece of email shows up, the URL is found and the new host name is looked up, and if the new name is found in the database, it is interpreted as spam. In addition to host names, a well-designed antispam database includes IP numbers.

To illustrate, consider a database that contains the host name `spam.example.com` and that host's IP number (192.168.33.44). When new mail arrives, the host name in the arriving mail is looked up in the database. If that new host also has the address 192.168.22.44, it too is rejected even though the new and old host names may be different.

2.8.1 THE HTML KEYWORD

The <a command always contains a web reference, but other HTML commands can also contain web references. The <a command indicates a web reference by using an `href=` expression. For example:

```
<a href="http://www.example.com">
```

But other HTML commands use different expressions to indicate the web reference. Table 2.1 lists the HTML commands that allow URLs and the expression used by each.

TABLE 2.1 HTML Commands That Reference URLs

Command	Expression	Description
<a	`href=`	Create a hyperlink (`href=`) or identifier (`name=`) in a document.
<applet	`codebase=`	Define an executable applet with a document.
<area	`href=`	Define a mouse-clickable area within a map.
<base	`href=`	The base for all URLs in the document.
<body	`background=`	Set background to a URL.
<del	`cite=`	Citation reference for deleted information.
<embed	`src=`	Embed an object in a document.
<form	`action=`	URL to use on submission of a form.
<frame	`src=`	Define a frame within a frameset.
<iframe	`src=`	Embed a frame inside a document.
<img	`dynsrc=`	Specify a video clip to play.

(continues)

TABLE 2.1 HTML Commands That Reference URLs *(Continued)*

Command	Expression	Description
<img	lowsrc=	Specify a low-resolution image to preload.
<img	src=	Specify an image to load.
<img	usemap=	Coordinates list for a map.
<ins	cite=	Citation for inserted commentary.
<input	src=	Image to select for an input choice.
<isindex	action=	Create a searchable document.
<link	href=	Create an interdocument link.
<link	src=	Specify an external style sheet to use.
<meta	url=	Reference for an HTTP refresh.
<object	classid=	Identify the class of an object.
<object	codebase=	Source of the code base for the object.
<object	data=	Source of data for the object.
<object	name=	Source for the name of the object.
<object	usemap=	Specify the image map to use with the object.
<q	cite=	Citation for the enclosed quotation.
<script	src=	Source for external language code to run.
<table	background=	Source of background image to load.
<td	background=	Source of background image to load.
<th	background=	Source of background image to load.
<tr	background=	Source of background image to load.

2.8.2 JUST IN CASE

Before digging deeply into URLs, we first need to comment on one of the (sometimes overlooked) characteristics of HTML in general.

First, note that all HTML commands and the URLs they reference are **case insensitive**. That is, all the following are the same:

```
<a href="www.example.com">
<A HREF="WWW.EXAMPLE.COM">
<a HrEf="wWw.ExAmPlE.CoM">
```

Note, too, that host and domain names are also case insensitive.

2.8.3 THE PROTOCOL SPECIFICATION

A **protocol** can be thought of as a language used by programs to communicate over a network connection.[10] Usually a request is sent from the client software to the server software, and an answer (or reply or data) is returned. The most common protocols are `http` (Hypertext Transport Protocol), `https` (HTTP with Secure Sockets Layer, or SSL), `ftp` (File Transfer Protocol), and `file` (for viewing local files).

In a URL, the expression that identifies the protocol prefixes the host or domain:

```
<a href="http://www.example.com" ... >
```

That prefix expression (here, the `http://`) is followed by a host or domain specification (or an IP number) and whatever additional information is needed:

```
<a href="http://www.example.com/cgi-bin/search?who=bob">
<a href="http://192.168.44.55">
```

Note that the protocol can be excluded, in which case it is automatically set to the document's default. The protocol, host or domain, and other information are normally enclosed in quotation marks to protect HTML-capable mail programs from stumbling over illegal characters. The quotation marks may be double or single, but whichever is used, they must pair up (a double quotation mark may not be mixed with a single quotation mark):

10. Some, such as `file`, interact with the local file system.

```
href="http://www.example.com"
href="http://www.example.com"
href="http://www.example.com"       Won't work.
```

Note, however, that quotation marks can often be safely omitted, so you should not count on their presence when parsing spam email.

If a spam HTML message is in a language other than English, the quotes may be present but specified using the other language's encoding:

```
href=EF2Dhttp://www.example.comEF2D
```

Here, the `EF2D` is hexadecimal that represents two binary byte values (not four characters) that specify quotation marks appropriate to the language. So again it is better not to depend on quotation marks when parsing URLs.

Although the protocol, when present, is always specified with a trailing ://, in actuality all that is really needed is the colon.[11] Thus, all three of the following produce the same URL result:

```
http://www.example.com
http:www.example.com
http:////////////www.example.com
```

Notice that the number of forward slashes is unimportant. The single required character is the colon.

Also note that there can be no space between the protocol and its colon, but the colon can be followed by arbitrary white space characters.

```
http :www.example.com       Space before colon won't work.
http: www.example.com       Space after the colon is OK.
http:                       A new line is OK.
www.example.com
```

11. We fudge for simplicity. Technically, two forward slashes are supposed to indicate a network path, and one forward slash is supposed to indicate an absolute (local) path (ref. RFC2396, section 3).

Note also that the protocol does not need to actually be present with each URL. A `<base` command (if present) sets a prefix that will precede all URLs that do not specify a protocol. The prefix is always terminated by a forward slash (/) even if one is omitted from the `<base` command.

```
<base href="http://www.example.com">
<img src="images/bob.jpg">
```

These two commands are the equivalent of the following (single) command:

```
<img src="http://www.example.com/images/bob.jpg">
```

If a `<base` is omitted and if the URL omits a protocol, the default is generally the `http://` protocol:

```
<a href="www.example.com">
```
The protocol defaults to http://.

2.8.4 EMAIL ADDRESSES MASK URLS

To protect from inappropriate input, some HTML-compliant mail readers interpret an email address that is part of an `http://` reference to be the same as a host or domain specification.[12] For example, the email address in the first line is interpreted (in the second line) as if the user part and the `@` were omitted:

```
<a href="http://bob@www.example.com">
<a href="http://www.example.com">
```

The lesson is that whenever you are parsing a host or domain specification, you will need to start parsing over again when you encounter an `@` character.

12. Technically, a mail reader discovers a *username:password* to the left of the `@` with the *:password* missing. Note that this form of address is actually useful for `ftp://` references (see RFC2396, section G.2).

2.8.5 IP NUMBERS TOO

Spammers are aware that the domain part of the URL does not need to be expressed in *host.domain* form (as `www.example.com`), and they use that fact to help disguise the host's name. For example, the following replaces the *host.domain* form with the IP number of the host `www.example.com`:

```
<a href="http://192.168.22.33">
```

Here `192.168.22.33` is the IP number for `www.example.com`. So be aware, when parsing URLs, that the *host.domain* part can be expressed as an IP number, too.

Also note that IP numbers can be expressed in decimal or in hexadecimal when prefixed with a literal `0x`, thus making them even harder to detect:

```
<a href="http://0xC0A81621">        IP number in hexadecimal
<a href="http://3232241185">        IP number in decimal
```

All this effort to disguise a host's name with a cryptic-looking hexadecimal address allows spam email to double as a means to accomplish fraud. Consider, for example, the following web reference and surrounding text:

```
Your ATM card PIN number has expired. For security
reasons, connect to our
<a href="https://www.ABCDE-Bank.com:Secure@0xC0A8162">
secure server</a> and select
a new PIN as soon as possible.
```

Users who click on this URL are taken to the spammer's fraud site at @0xC0A8162, and not to the bank's site as they would expect. Even more dangerous, users see the following literal link address in the browser's link window, thereby being further fooled into thinking that the link is legitimate:

```
https://www.ABCDE-Bank.com:Secure@0xC0A8162
```

This example shows why it is crucial to start over when you encounter an `@` when parsing a URL and why you need to allow for *host.domain*, IP address, and hexadecimal forms of addresses when parsing the URL.

2.8.6 DEALING WITH REDIRECTS

A **redirecting site** is a host that takes a web reference that points to itself, strips away the self-referencing part, and then issues an HTTP redirect command back to the user's browser with the remaining part of the reference. The effect for the user is to view the redirected-to site and not the redirecting host's site.

```
http://redirect.example.com/*http://www.real.host
http://redirect.example.com/*http://www.real.host
http://www/real.host
```

Here, `redirect.example.com` is a redirecting site. When a browser visits it with the full URL shown on the first line, it recognizes the self-referencing part (bold on the second line), strips that self-reference, and returns an HTTP redirect to the actual host (shown on the third line).

The presence of redirect sites increases the complexity of parsing an actual site from a URL in spam email. Currently it is sufficient to simply start parsing over again when you encounter an `http:` or an `https:` while parsing a URL. As spammers gain skill and experience, however, such a simple solution will become less effective.

There are many redirect sites on the Internet. One that spammers know well is rd.yahoo.com.[13] They all share the same characteristics. The URL for the redirecting host is listed first, followed by a forward slash or backslash, then a special character, and then the URL for the actual host.

```
<a href="http://rd.yahoo.com/xxxx/*http://real.site">
```

For the rd.yahoo.com family of servers, the special character is an asterisk. For others the special character varies. The one common characteristic among all redirect servers is that the special character is not one that would normally appear as part of a URL.[14]

The redirect site can be followed by an arbitrary amount of URL information. For example:

```
href="http://rd.yahoo.com/bypass/winkie/food/thumb*big/dairy/
noisy/gyroscope/middle/fred/*http://real.site"
```

13. Another is g.msn.com.
14. Don't be surprised if a spammer organization forms to host a redirect site that uses a special character that can appear in a URL.

Recall that the special character must follow a forward slash. So here the * in `thumb*big` is not special. That is, it does not begin a URL for the real site.

Note that backslashes are the equivalent of forward slashes in the redirect portion of the URL because they are essentially ignored by the redirect server:

```
href="http://rd.yahoo.com\bypass/winkie/food\thumb*big/dairy/
noisy/gyroscope/\middle/fred/http://www.chunky.example.com/
acidrain/moose/jane/wheel\*http://real.site"
```

Here, the actual site (always last) is `real.site`. But beware: `real.site` is the spammer's site and subject to the spammer's rules. It is not unreasonable to expect spammer sites to employ new methods that will make the `real.site` appear perhaps second from the last or even third from the last:

```
href="http://rd.yahoo.com\bypass/winkie/food\thumb*big/dairy/
noisy/gyroscope/\middle/fred/http://www.chunky.example.com/
acidrain/moose/jane/wheel\*http://real.site?search=http://
some.bogus.site&referenece=http://another.bogus.site"
```

Just as it is important to begin parsing over again when you find an `http:` or `https:`, it is also necessary to stop parsing when a URL argument (the `?`) appears. Clearly, spammers will go to great lengths to disguise the actual URL, and the examples we have shown here are only a hint of the techniques you will see in the future.

2.8.7 WILDCARD DNS RECORDS

The host name part of a URL can be subject to the same random word masquerading as the body, making it difficult to detect. To illustrate, consider these two URLs:

```
href="http://bob.biff.bonny.bill.betty.boop.example.com"
href="http://andy.able.alex.annie.alice.boop.example.com"
```

If you look at only one of these, there is no way to know for certain where the randomizing ends. But looking closely at two, one might surmise that the host name is `boop.example.com`. But one might be wrong. The real host might actually be `example.com`. To find out which is right, we need to delve briefly into DNS records.

When an HTML-capable mail program needs to connect to a URL's site, it first must look up the address of that site. For both of the sample randomized host names, it would (for example) find the address 192.168.111.44. But look at what happens when we look up the two names we suspect to be the real host names for these URLs:

```
boop.example.com    →   192.168.1.23
example.com         →   192.168.111.44
```

Clearly, our presumption—that the `boop.example.com` host was the real host—was wrong. Instead, the real host is `example.com` because it has the address 192.168.111.44.

But a savvy spammer might anticipate this logic and use a host name that appears random but is actually the real host name.

```
boop.example.com    →   some innocent's address
example.com         →   some innocent's address
able.alex.andy.boop.example.com  →   192.168.111.44
```

Here, if you decide to use the address that you suppose is the real host, you might cause some innocent site's address to be interpreted as that of a spamming site. The correct way to record the spamming address for later use is to look up and record the full domain name in the reference, even if it appears to be random.

2.8.8 CNAME RECORDS AND URLS

When an HTML-capable mail program attempts to look up a URL's host or domain, it may receive another host name in return rather than the expected address. That new host name is called a CNAME record. Here's how it works:

- You find the URL `http://example.com`.
- You look up the host name `example.com`.
- You expect an address, such as 198.162.33.44, but instead you get the host name `www.example.net`.

When you look up a URL's host name and expect an address but instead get another host name in return, you need to do an additional lookup to find the actual address.[15]

15. We fudge for simplicity. Sometimes the new host and its address are returned at the same time.

```
    example.com  →   www.example.net
www.example.net  →   192.168.33.44
```

But CNAMEs can lead to other CNAMEs, thus creating a long thread of potential lookups, and CNAMEs can even form infinite loops:

```
    example.com  →   www.example.net
www.example.net  →   www.example.com
www.example.com  →   example.com       An infinite loop!
```

When you combine the risk of CNAMEs with the need to decipher long host names, the thread of lookups might get ugly indeed:

```
able.alex.andy.boop.example.com  →   www.example.net
               www.example.net  →   boop.example.com
              boop.example.com  →   192.168.111.44
    alex.andy.boop.example.com  →   192.168.22.4
         andy.boop.example.com  →   10.9.4.2
              boop.example.com  →   192.168.111.44
                   example.com  →   example.net
                   example.net  →   example.com
                   example.com  →   example.net
                   example.net  →   example.com
                        etc. in an infinite loop
```

When you write code to decipher a long host name, be sure to account for the possibility of infinite loops.

2.8.9 URLs Used as Comments

Just as non-HTML words can be used to create comments (see section 2.5), so can URLs. For example, consider the following:

```
V<a href="bob.example.com"></a>i<a href="jane.example.com></
a>a<a href="alice.example.com"></a>g<a href="dan.example.com"></a>ra
```

When viewed on an HTML-aware mail reader, the preceding would appear like this:

```
Viagra
```

The use of URLs as comments is intended to make it difficult to find the actual URL in the message. When you look for the URL, it is not enough to simply look for the abutting the reference, because arbitrary nonprint HTML can also appear between the two:

```
V<a href="bob.example.com"><font size="+5"></font></a>i<a
href="jane.example.com"><font color="white"></font></a>
```

Finding the URL when this technique is used requires your spam scanner to act almost as an actual HTML parser.

2.8.10 JAVASCRIPT.ENCODE URLS

Last here—but certainly not the last word in hiding the URL—is the technique of encoding a URL using JavaScript.Encode. The spammer's idea in this strategy is to wrap the URL in JavaScript so that it will be decoded by the browser. For example, consider the following obscured URL (wrapped to fit the page):

```
<script language="JScript.Encode">#@~^hQAAAA=3D=3D~@#@&[Km!:+
YcADbYn'E@> !(o"bHA~?"Z'r4OYa)Jz+!+ O, FF+R8*f&^kxV
4YhVr~qq9:C{!P_2&!C:'TPwI)\AAr"92"> '!,j/I}SdqHMxE
WE@*@!&qwI)\A@*BbI@#@&AyIAAA==^#~@</script>
```

Here, the HTML tag, <script, tells the interpreter to decode what follows using the JavaScript.Encode protocol. That protocol will then decode everything between the leading <script and the ending </script>. When decoded, the URL becomes the following:

```
document.write('<IFRAME SRC="http://192.168.23.45/link.html"
WIDTH=0 HEIGHT=0 FRAMEBORDER=0 SCROLLING=0>')
```

After decoding, it is clear that the encoded URL contains a web reference that the spammer wants to keep secret. But we can use it to record the URL of the spamming site. Here, that site is represented by the IP address 192.168.23.45, but in other JavaScript. Encoded URLs the reference may be a host name or may be further obscured by other means.

See http://www.virtualconspiracy.com/ for C language source examples of ways to decode JavaScript.Encode.

2.9 LOOSE ENDS

Most spam email is trying to sell something to the recipient and thus needs to entice the recipient into contacting the spammer. As you have seen, finding a website buried inside spam email can be tricky. But the effort is worthwhile because most spam email sells by tricking or enticing the user into clicking on a website.

Some spam omits a web reference and instead provides a phone number. Because there is no universal standard for the appearance of phone numbers, detection of phone numbers is more an art than a science. Consider, for example, the following ways to hide phone numbers:

```
Call now! 1 8 0 0 (yes the call is free!) 555 12 12
Dial one eight-hundred 555 1234
Ring us at 800 LLL-LUCK
```

Fortunately, the use of phone numbers is rare (probably because spammers don't want to give out their phone numbers). But if a URL is missing and if a phone number is easy to find (that is, if it is not an image), look for a phone number instead.

Even more rare, instead of a URL you will find only an email address. On those rare instances when only an email address is available, record it.

Note that some spam doesn't want you to connect at all. It merely recommends a stock or a product and hopes the recommendation will cause you to buy. Obviously this type of spam is very difficult to detect the first time detection is attempted. But note that its pattern can later be recognized as spam once a human has categorized it, as with a Bayesian filter (see section 11.8).

Finally, of course, a tiny amount of spam is not motivated by sales at all. Some is religiously or politically motivated. During one month, for example, the authors were spammed with dozens of requests to read and live by a psalm in the Bible, where the spam recommended a different psalm in each mailing. Such altruistic spam is the hardest of all to detect.

2.10 THINK LIKE A SPAMMER

Spammers are not creatures apart from the rest of humanity. They are human, just like you and us, and are motivated just the same, to make a reasonable living.

Imagine yourself out of work, desperate and needing to make a buck. All you have is your computer, lots of computer savvy, and a money-making scam in mind. What would

you do to send spam email? What would you do that is different from what spammers are doing now? How would you try to foil current spam detection software?

By trying to outspam the spammers, you will be better prepared to fight them when they later mature to your level of thinking.