

Anonymous Mail Forwarding Vulnerabilities in FormMail 1.9

*Ronald F. Guilmette <rfg@monkeys.com> - Infinite Monkeys & Co.
Justin Mason <jm+fma@jmason.org> - maintainer of SpamAssassin*

Sun Jan 23, 2002

Introduction

This advisory describes various methods and mechanisms for the remote abuse of the widely-used FormMail script, version 1.9, to send arbitrary e-mail messages to arbitrary recipients without the consent of, and against the wishes and intentions of those who have installed the script on their web servers. Essentially all of the exploits described herein are also applicable to earlier (pre-1.9) versions of FormMail.

General information on the widely-used free FormMail CGI script may be found at the following URL:

`http://worldwidemart.com/scripts/formmail.shtml`

Impact

By manipulating inputs to the FormMail CGI script, remote users may abuse the functionality provided by FormMail to cause the local mail server on the same (web) server system to send arbitrary e-mail messages to arbitrary e-mail destination addresses. Such e-mail messages may contain real or forged sender e-mail addresses (in the From: headers) entirely of the attacker's choosing. In some cases, the envelope sender addresses of such messages may also be set to arbitrary values by the attacker.

When and if the vulnerabilities described below do exist (which is dependent, in some cases, upon script installation configuration choices and web server configuration choices) and when and if they are in fact exploited by an outside attacker, message recipients will have only incomplete e-mail tracing information available in the message e-mail headers. This incomplete tracing information will lead back only as far as the web server that hosts the FormMail script. The IP address actually used to initiate the messages will not be available to these recipients. The originating IP address of the attacker may or may not be available (in local web server logs) to the administrator of the exploited web server, depending on local web server logging, log file retention policies, and the length of time that elapses between exploitation and the time the local administrator becomes aware of the exploitation. (Of course, even in cases where web server logs are available, tracing information contained therein may perhaps allow tracing of the attacker only as far back as whatever anonymizing HTTP proxy the attacker used.)

Notification

We have followed the notification guidelines laid out in:

<http://www.wiretrip.net/rfp/policy.html>

with respect to this advisory. Both authors of the present advisory attempted to contact the FormMail maintainer via the e-mail address given on the FormMail home page more than five working days prior to publication of this advisory. No response from the maintainer was forthcoming. Attempts were also made to contact the maintainer via telephone but were unsuccessful.

Background

The FormMail script has a long and unenviable history with regards to security issues. Originally intended as a helpful free CGI script, usable in the construction of simple interactive 'contact us' type web forms, it was originally created and distributed July 9, 1995 and subsequently underwent several revisions culminating in version 1.6 which was released May 2, 1997. The 1.6 version became widely used, and copies of the 1.6 version are still installed and publicly accessible in many locations at the present time.

Sometime between the release of the 1.6 version of FormMail and March, 2001, various Internet users became aware that spammers were exploiting the relatively modest and weak security checks within the 1.6 version to send large quantities of unsolicited 'spam' e-mail to large numbers of recipients via exploited FormMail scripts. In effect, spammers were using the exploited FormMail scripts in ways that rendered them functionally equivalent to anonymizing open e-mail relay servers. Eventually, someone filed a public security advisory regarding these exploitations and the now evident security problems with the FormMail script.[1] Later reports indicated a belief that spammers were actively searching the net for exploitable FormMail scripts.[2] At least two different parties have apparently been motivated (by FormMail's evident security problems) to create and distribute their own independently-developed versions of FormMail which allegedly close the security loopholes.[3]

Subsequent to the release of the 1.6 version of the script, 1.7 and 1.8 versions were also released. On his web site, the FormMail author himself notes that these versions have unspecified (and perhaps different; see footnotes) security vulnerabilities, and that all users should upgrade to version 1.9 "IMMEDIATELY".

Version 1.9 of FormMail was released by its author on or about August 3, 2001, according to the comments in the script. This version has been alleged to rectify the various previously identified security issues with FormMail, including its prior ability to be manipulated into behaving like an anonymizing open mail relay.[4]

Pre-1.9 Exploitation

Exploitation of FormMail versions prior to 1.9 for the forwarding of e-mail messages which have been effectively *anonymized*[5] was rendered trivial by the script's total reliance on information transmitted by the web client to the web server (and hence to the FormMail CGI script) for both authentication of the request and for specification of recipient e-mail address(es) for each message.

FormMail's authentication of incoming requests, such as it was (and is, in version 1.9), is limited to the following rather rudimentary check on the `HTTP_REFERER` environment variable passed to the script from the local web server. (The web server, in its turn, obtains this value from the `Referer`: HTTP header supplied by the HTTP client.)

```
if ($ENV{'HTTP_REFERER'}) {
    foreach $referer (@referers) {
        if ($ENV{'HTTP_REFERER'} =~ m|https?://([^\/*]*)$referer|i) {
            $check_referer = 1;
            last;
        }
    }
}
else {
    $check_referer = 1;
}
```

Note that further up in the script, the original script installer is required to have defined the `@referers` array to a list of strings, each of which is supposed to be a domain name or IP address. These domain names and/or IP addresses are, by their inclusion in the installer-supplied definition of `@referers`, effectively authorized to run web servers that will serve up HTML pages containing HTML forms that may work with this specific locally-installed copy of FormMail. So, for example, if the local FormMail installer had wanted to limit the script's usage so that it could only be used in conjunction with HTML forms resident on the web servers known as `www.example.com` and `www.example2.com` then the installer would have previously defined the `@referers` array as follows:

```
@referers = ("www.example.com", "www.example2.com");
```

There exist several ways to easily bypass and circumvent the `HTTP_REFERER` checking code shown above. The most obvious way, and the way that most attackers will certainly use is simply to decline to provide any `Referer`: header at all as part of the HTTP request. In such cases, the attacker will effectively “pass” the `HTTP_REFERER` validation test[6], and the remainder of the FormMail script will then be executed.

Even though it is clear that the simplest and easiest way to circumvent the `HTTP_REFERER` check in FormMail is simply to avoid including any `Referer`: header in the HTTP request,

for the sake of completeness we would also like to note some other methods whereby this same check could be circumvented even in programming contexts[7] where omission of `Referer:` headers may be difficult or impossible to achieve.

Specifically, it should be noted that the Perl regular expression matching attempt:

```
$ENV{'HTTP_REFERER'} =~ m|https?://([^\ ]*)$referer|i
```

is not left-anchored prior to the `https?:` scheme specification nor is it right-anchored after the `$referer` domain name.

The lack of left-anchoring prior to the `https?:` scheme specification portion of the regular expression allows an attacker to “pass” this simple-minded `HTTP_REFERER` validation test as long as the `HTTP_REFERER` value (set from the `HTTP Referer:` header) is some URL containing a match for the regular expression anywhere within its entire length.

An attacker may easily exploit this lack of left-anchoring by simply loading, and then submitting an HTML form to the desired FormMail CGI from the attacker’s own local web server using an extended ‘spooF’ URL which contains a substring matching the regular expression somewhere within the entire URL string. The matching substring may be presented as part of the right context of the complete URL, and specifically within a portion of the URL that will, by convention, be ignored by many/most web servers when loading a page corresponding to the given URL. For example, the following URL:

```
http://www.attacker.tld/myform.html?http://www.victim.com/
```

if included as the URL in an `HTTP GET` request will typically result in the HTTP client receiving a copy of the same web page that would have been obtained if the requested URL had been just:

```
http://www.attacker.tld/myform.html
```

because a typical web server will simply ignore the trailing portion of the URL past the question mark.[8] Typical HTTP clients on the other hand, if asked to perform such a `GET` operation, will do so (with the help of the relevant web server) but will then preserve the entire original request URL and will later present that as the `Referer:` value, when and if some HTML form on the page is the subject of a submit operation.

The HTTP client’s preservation of the entire URL, including the trailing ‘spooF’ substring, allows most typical HTTP clients to trivially pass FormMail’s simple-minded `HTTP_REFERER` validation test even if the attacker has neither built nor obtained any HTTP client software that allows for suppression of `Referer:` headers[9], and even if he is writing his own attack software in some language (e.g. Javascript) that otherwise makes the implementation of attacks on web CGIs particularly easy and convenient.

The lack of right-anchoring after the `$referer` portion of the regular expression may also be exploited by an attacker wishing to create a modified FormMail HTML submission form on a web server of his own choosing. This may be more difficult than exploiting the lack of left-anchoring however, and may perhaps only be achievable with the aid of some cooperative DNS server. Given the availability of such a server however, a DNS 'A' record could easily be defined for an fully-qualified domain name such as:

```
www.victim.com.attacker.tld
```

and that 'A' record could be defined with an IP address of the attacker's choosing. In this case, the attacker could easily load his malevolent locally-installed HTML exploit form (which has its `ACTION=` clause set to point to the victim's FormMail) into his browser via a URL such as:

```
http://www.victim.com.attacker.tld/myform.html
```

The attacker could then simply submit the form and it would 'pass' the simple FormMail `HTTP_REFERER` validation check.

Note that all three methods for circumventing FormMail's `HTTP_REFERER` validation remain present in the current (1.9) version of the script, i.e. (1) complete omission/suppression of `Referer:` headers and (2) exploitation of the lack of right-anchoring and (3) exploitation of the lack of left-anchoring.

Once an attacker has passed the easily circumventable `HTTP_REFERER` validation check, he then has a mostly free hand to manipulate other CGI (form input) variables in order to direct an e-mail message of his choice to some selected destination address or addresses.

FormMail obtains the desired destination e-mail address for each transmitted e-mail message from the HTTP client. The HTTP client, in turn, is expected to obtain it from the pre-defined value of a *hidden* HTML form field (`recipient`) within a form (or forms) associated with the specific web site/server where a given instance of FormMail is installed. Obviously however, if (as we have seen above) it is trivial to trick FormMail into believing that some attacker-created form which actually resides on some arbitrary web server of the attacker's choosing is a valid *referrer*, then attackers can certainly manufacture (or simulate) forms with the `recipient` form field set to any e-mail address desired.

These are the conditions which led to earlier reports of FormMail being exploited, generally by bulk e-mail spammers, as if it were a kind of *anonymizing open e-mail relay server*. The fundamental problem was that the `HTTP_REFERER` checking could be easily spoofed, and that the pre-1.9 versions of FormMail performed no validation checking of any kind on the `recipient` CGI parameter. Attackers were thus allowed to set this parameter to any desired value.

Version 1.9 of FormMail incorporated changes whose purpose was to allow the actual destination e-mail addresses (i.e. the value given for the `recipient` CGI parameter) for any submitted message to be restricted by checking them against an installer-supplied list of "acceptable"

recipient e-mail addresses and/or recipient domains. As noted below however, these changes did not fully achieve their goal. That unfortunate fact, together with the multiple flaws in `HTTP_REFERER` checking (all of which are still present in the 1.9 version) imply that even the current 1.9 version of FormMail can still be used and abused for spamming, for e-mail harassment, or other purposes neither condoned or intended by the FormMail installer.

Anonymous Mail Forwarding Exploits in FormMail 1.9

As noted above, versions of FormMail up to and including 1.8 are believed to be vulnerable to various security exploits, including but not limited to creative generation of HTTP header and message data, leading to an installed FormMail CGI script at a targeted site behaving like an anonymizing open e-mail relay. At present, spammers are continuing and increasing their abuse of such earlier versions of FormMail, especially the widely deployed 1.6 version, at various sites where these earlier versions are, unfortunately, still installed.[10]

This advisory describes various vulnerabilities in version 1.9 of the FormMail script that may also lead to misuse and abuse of the script by remote attackers to achieve an effect functionally equivalent to an anonymizing open e-mail relay server.

Note that many, if not most of these exploits were also present in versions of FormMail prior to the 1.9 version, and thus it may be fairly said that the exploits described herein are not in any sense new to the 1.9 version. For the sake of brevity however, we report these problems as vulnerabilities in the 1.9 version.

The vulnerabilities described herein may be broadly separated into the following categories: (1) creative recipient addressing, (2) exploitation of the `email` and `realname` CGI parameters, (3) other regular expression matching issues, and (4) mail-bombing by proxy using FormMail. These vulnerability categories are each described in separate sections below.

Creative Recipient Addressing

The preceding section noted that FormMail 1.9 suffers from the same `HTTP_REFERER` validation flaws as earlier versions of the script, thus forcing the prevention of unauthorized use/misuse of the script to rely entirely on the code added in 1.9 which attempts to validate client-supplied recipient e-mail addresses. These added checks contain a number of subtle and not-so-subtle flaws however, any one of which, if exploited, renders the checking code useless and ineffective.

Briefly, the recipient address checking code added in 1.9 merely attempts to ensure that each comma-separated substring of the client-specified `recipient` string matches (with right-anchoring only, and without case sensitivity) one of a set of strings specified by the person who installs the FormMail script. The array called `@recipients` is used to hold these (valid recipient address or domain) strings and each comma-delimited substring of the client-supplied `recipient` CGI parameter string is checked to see that it matches (right-anchored) at least one of these installer-specified “approved” recipient address or domain strings.

As long as this check ‘passes’, for any given comma-separated substring of the `recipient` value, that substring will be saved in yet another array (`@send_to`), and later on, all elements of the “validated” `@send_to` array are join’d back together, with comma separators, and the resulting string replaces the old, unvalidated `recipient` CGI parameter value. That value is then later used (when writing e-mail headers to the local mail server) in the following Perl statement:

```
print MAIL "To: $Config{'recipient'}\n";
```

So, for example, if the installer of the script had provided a definition of `@recipients` like:

```
@recipients = ('pres\@elsewhere.tld', 'ourdomain.tld');
```

then all of the following client-supplied recipient addresses, if given by the HTTP client, either alone or as elements of a comma-separated list of recipient addresses, would “match” some element of the `@recipients` array definition shown above, and would thus be allowed as valid recipient addresses:

```
pres@elsewhere.tld
PRES@elsewhere.tld (See note #1)
vice-pres@elsewhere.tld (See note #2)
Professor.Plum@ourdomain.tld
Norma.Desmond@nostalgia.ourdomain.tld
user%somewhere.tld@ourdomain.tld (See note #3)
user@somewhere.tld(ourdomain.tld (See note #4)
user@somewhere.tld(pres@elsewhere.tld (See notes #2, #4)
<user@somewhere.tld>ourdomain.tld (See note #5)
<user@somewhere.tld>user@ourdomain.tld (See note #6)
<user@somewhere.tld>pres@elsewhere.tld (See notes #2, #6)
```

Unfortunately, some of the “acceptable” strings shown above, if crafted by an attacker and then used within a spoofed/fraudulent HTTP request may lead to undesirable consequences, i.e. the forwarding of an associated e-mail message to some recipient (or recipients) having e-mail addresses that the installer never intended to allow as FormMail message recipients.

Several notes regarding the example strings shown above are in order:

1. Comparison of each comma-separated sub-part of the client-supplied `recipient` string is performed without case sensitivity. On systems where the initial (user ID) portion of e-mail addresses is considered case-sensitive, this may allow misuse of FormMail to send e-mail messages to unintended recipients having e-mail addresses with alternative case on the same system. Worse however, this possibility opens up a hole that might perhaps allow a remote attacker to misuse an installed FormMail script in a way that avoids immediate detection by any local user or administrator. We will return to this point below.

2. In fairness to the FormMail author, the README file supplied with FormMail version 1.9 explicitly and strongly suggests that installers should prefix any and all full e-mail addresses they include within the definition of the `@recipients` array with the up-caret (^) character, thereby causing subsequent regular expression matching for those strings to be anchored both on the left and on the right. Following that suggestion would certainly eliminate the potential exploits noted above that refer to this note. Anyone following the FormMail author's advice with regard to 1.9 installation would have written their `@recipients` definition as follows:

```
@recipients = ('^pres\@elsewhere.tld', 'ourdomain.tld');
```

rather than the way it was written further above, thus eliminating such obvious potential problems as being able to trick FormMail into sending unauthorized e-mail messages to `vice-pres@elsewhere.tld`. As we have already noted however, even when each and every element of the installer-defined `@recipients` array is set to a full e-mail address (as opposed to merely a domain name) and even when each of the e-mail addresses in question is left-anchored, the check for correspondence between client-supplied recipient addresses and the installer-defined set of allowed recipient addresses is performed in a case insensitive manner. We will return to this issue below.

3. Although the `recipient` parameter provided to FormMail by the HTTP client will have newlines and carriage return characters, if present, removed by FormMail, and although the `recipients` string is separated into a set of comma-separated substrings before the recipient address validation occurs, no other validation of the form, format, or syntax of the recipient address(es) is performed. This leaves open the possibility of using the old and widely-known Sendmail *percent hack*[11] form of e-mail addressing, at least in cases where one or more of the elements of the `@recipients` array are just domain names, as opposed to full e-mail addresses.

To understand fully why use of the percent hack is almost guaranteed to work in such cases, one must understand that:

- FormMail is designed to work in conjunction with either Sendmail or with some work-alike program, such as Postfix, and we may thus assume that whatever mail server FormMail is hooked up to will almost certainly be one that *does* support the *percent hack* form of addressing.
- In normal (non-exploit) operation, the mail server resident on the same host as the FormMail script itself will (unless the administrator has botched either his Sendmail configuration or his FormMail configuration) accept all “locally generated” e-mail messages coming out of the FormMail script and addressed to any e-mail address whose `@domain` part is the same as of any one of the domains listed in the `@recipients` array. Thus, in the preceding examples, we can usually be sure that the local mail server on the FormMail host will not disallow or reject any such “locally-generated” e-mail message that is addressed to `user%somewhere.tld@ourdomain.tld` because it will see that the domain portion of this address is `ourdomain.tld`. That alone will probably be

enough to satisfy Sendmail that it should accept (and, if necessary, forward) the mail message in question. Also, a typical Sendmail installation, even a modern one, will usually not itself object to any kind of forwarding of any “locally-generated” e-mail message to any other domain or site, because it is typical to assume (when configuring Sendmail) that “local” e-mail originators can be fully trusted, at least to send outgoing e-mail to other sites. (Note that FormMail is acting in the role of a “local” and *trusted* user in this case.) So an attacker really only needs to get past FormMail’s check on HTTP_REFERER (which, as we have seen, is trivial to do) and then, if he can also get past FormMail’s check of the right-most part of the recipient address string(s) against the @referers array he will be able to send e-mail messages to essentially any e-mail address anywhere on the Internet. As illustrated by the example above, use of the *percent hack* form of e-mail addressing allows the attacker to get past the latter check.

4. RFC 2822, like RFC 822 before it, specifies that within the text of so-called *structured* mail headers (e.g. To:) any string enclosed within matching left and right parenthesis shall be taken as being purely commentary material, and shall be ignored. Thus, if an attacker provides a recipient string of the form:

```
user@somewhere.tld(ourdomain.tld
```

and if that string passes the (right-anchored) regular expression matching check (i.e. matching at least one element of the @recipients array), then a To: header line like:

```
To: user@somewhere.tld(ourdomain.tld
```

will be generated by FormMail and passed to Sendmail. In such a case, Sendmail should take everything between the left parenthesis and the corresponding right parenthesis as a comment, thus leaving it with only the user@somewhere.tld address to act on. However in this case, there is no closing right parenthesis before the end of the header line, so this would appear to be a case where the actual behavior of the mail server cannot be predicted. In practice however, both Sendmail and Postfix helpfully and implicitly supply the missing closing right parenthesis themselves, with the result being that the (ourdomain.tld portion of the To: header is in fact ignored, leaving only the user@somewhere.tld part to define the actual e-mail recipient address.

5. For a client-supplied recipient string of the form:

```
<user@somewhere.tld>ourdomain.tld
```

Sendmail will subsequently be fed the corresponding mail header:

```
To: <user@somewhere.tld>ourdomain.tld
```

and when such a header is supplied to Sendmail along with Sendmail’s -t (take recipient

address(es) from message headers) option, reports indicate that Sendmail will send a copy of the message to `user@somewhere.tld`, and that it will then ignore the remainder of the header line, i.e. the part past the right (closing) angle bracket. (This is in fact entirely consistent with what the mail header parsing rules contained in RFC 822 and 2822 would indicate *should* actually happen in such a case.)

6. For a client supplied recipient string of the form:

```
<user@somewhere.tld>user@ourdomain.tld
```

it would appear clear that if `ourdomain.tld` is included in `@recipients`, then such a string will pass FormMail's recipient address acceptability check, and later, when Sendmail is fed:

```
To: <user@somewhere.tld>user@ourdomain.tld
```

then Sendmail will most certainly send a copy of the associated e-mail message at the very least to `user@somewhere.tld`, but possibly also to `user@ourdomain.tld`. Reports suggest however that Sendmail will "parse out" the latter e-mail address and send the message only to the former address. Even if the local mail server used on a given victim server is one that will parse the header shown above and then send to both e-mail addresses, an attacker could potentially choose some infrequently- or never-monitored e-mail address (such as the traditional `nobody` user ID) within the targeted domain and use that e-mail address to provide the right context for his (spoofer) recipient string. That also will allow him to get past FormMail's recipient address acceptability checks.

Exploitation of `email` and `realname` CGI Parameters

As noted in the preceding section, clever manipulation of the `recipient` CGI parameter when invoking FormMail 1.9 can allow an attacker to bypass the recipient address checks imposed by the installer's definition of the `@recipients` array, at least in cases where the installer has failed to define elements of this array to complete e-mail addresses (i.e. when one or more elements of the array is just a domain name) and when and if the installer has failed to left-anchor (with `^`) each full e-mail address within the `@recipients` array definition. Even in cases where all elements of the `@recipients` array are full e-mail addresses and where these are all left-anchored, an attacker may perhaps still be able to bypass FormMail's recipient address checks by exploiting various weaknesses in FormMail's regular expression matching code, as we will show in the next section.

Clever bypassing of the checks applied to the comma-separated substrings of the `recipient` CGI parameter may however not even be necessary so long as the attacker is willing to supply cleverly constructed values for the `email` and/or `realname` CGI parameters, and so long as the attacker is willing to allow one copy of his e-mail message to go to some "permitted" recipient address, in addition to some other attacker-specified list of addresses.

The `email` and `realname` CGI parameters may undergo even less scrutiny by FormMail than the comma-separated `recipient` substrings. In fact the `email` and `realname` CGI parameters may undergo no validation whatsoever. (The `realname` parameter most certainly does not undergo any validation whatsoever within FormMail, and the `email` parameter will not undergo any validation as long as it is not listed as a *required field* in the list of required form fields that is itself initialized from the client-supplied value of yet another hidden form field. Given that an attacker, either a determined one or a hurried and careless one, will most probably set the value of the `required` hidden form field to null or no value, we will just simplify the remainder of our presentation by assuming, from here forward, that neither the `email` CGI parameter nor the `realname` CGI parameter undergo any validation.)

One critical additional point to note here is that neither the `email` CGI parameter value nor the `realname` CGI parameter value are in any way “cleaned up” by FormMail before they are printed to the local mail server. Specifically, the values of these variable do not have carriage returns or newlines removed before they are given to the mail server.

Once they have been supplied by the HTTP client, both the `email` and `realname` CGI parameter values are printed, verbatim and without filtering, to the local mail server via the following Perl statement:

```
print MAIL "From: $Config{'email'} ($Config{'realname'})\n";
```

One’s first thought upon seeing this statement is that the lack of filtering or validation applied to the `email` and `realname` CGI parameters cannot possibly be harmful, given the benign context (i.e. a `From:` header) within which these values will appear. That thought would be misleading however, because of the lack of newline removal for these parameter values.

Because an attacker can embed newlines in either or both of these CGI variable values, he can also effectively change the *context* in which the remaining (rightward) characters of either value appear to the local mail server. Specifically, an attacker could set the value of the `email` parameter to some string such as:

```
IGNORED\nCc: user@somewhere.tld,...
```

where `\n` represents a newline character. This rather simple trick provides the attacker with access to a self-generated `Cc:`, `Bcc:`, or (secondary) `To:` header context which, once achieved, provides the attacker with an avenue to direct the local mail server to send the attached message to some arbitrary set of e-mail addresses, in addition to whatever address may have been specified (via the `recipient` CGI parameter) for inclusion in the earlier `To:` header.

As should be apparent from the Perl `print` statement shown above, this same context shift (achievable via embedded newlines) may also and equally be exploited via clever construction of a suitable attacker-supplied value for the `realname` CGI parameter. In this case however, the attacker would need to take care to also escape from the confines of the paired parentheses that the script will supply to enclose the `realname` value. Given that the `realname` parameter

value will appear in the mail headers surrounded by a pair of open and close parenthesis, the “smuggling in” of additional recipient addresses via the `realname` CGI parameter requires the attacker to include some strategically-placed parenthesis of his own in the value he passes to the victim web server for the `realname` parameter. For example, the value:

```
IGNORED)\nCc: user@somewhere.tld\nX-Ignore: (
```

could be supplied by an attacker as the `realname` CGI parameter value, thus closing the already-opened SMTP header comment, escaping to, and creating a new (`Cc:`) e-mail header context (which is then filled with victim e-mail addresses), escaping again to another new e-mail header context, and then finally supplying a open parenthesis to match the closing one that will be supplied by the Perl print statement within FormMail.

Last but by no means least, it should be noted that vulnerable FormMail installations may be less than ideal as a vehicles for distributing e-mail messages, either unsolicited or otherwise, due to FormMail’s rather inconvenient (for the spammers) habit of appending its own unique legend just ahead of the body text of any given mail message that it forwards. FormMail always prepends a set of body text lines having the following general form to each e-mail message it passes to the local mail server:

```
Below is the result of your feedback form.  It was submitted by  
(sender address) on date/time stamp
```

This additional leading body text would appear to be annoying to spammers who would much prefer to have the text of their own messages appear at the very start of the message body text. (We assume that this spammer-annoying aspect of FormMail is one of the reasons, if not the primary reason that spammers have generally shown a preference for using open mail relays, as opposed to open FormMail scripts, for sending their messages.)

Unfortunately however, given that the `email` and `realname` CGI parameters of FormMail may contain embedded newlines, it is apparent that either of these parameters may be set, by an attacker, to a value which includes not only additional recipient addresses, but also additional mail headers and additional initial message body text of the attacker’s choosing. For example, the `email` parameter could be set to:

```
\nCc: user@somewhere.tld\nSubject: $$$\n\nMAKE MONEY FAST!!!
```

thus causing the text "MAKE MONEY FAST!!!" to appear in the outgoing message as the very first line of the message’s body text. Furthermore, a value such as the one shown just above could be followed by arbitrarily many additional newline characters, thus pushing the FormMail supplied legend text far down below the attacker’s desired subject header and initial message text (and perhaps even entirely off the ‘front page’ of what the messages recipient will see when he first looks at the received message).

This tactic may of course be applied with essentially equal effect to either the `email` or `realname` parameters, except that the caveats mentioned above concerning the balancing of parenthesis would apply in the case of manipulation of the `realname` parameter.

These possible tactics for “hiding” the otherwise spammer-annoying FormMail leading body text are not, unfortunately, the only ones possible. As one careful early reviewer of this advisory noted, the vulnerability which allows an attacker to include newlines in the `email` and/or `realname` CGI parameters might also open up an avenue whereby the FormMail-supplied leading body text could be effectively removed from the view of a message recipient via the careful construction of MIME e-mail headers which achieve that exact effect. (This design of an exploit making use of this approach is left as an exercise for the reader.)

Other Regular Expression Matching Issues

We have noted above several ways of bypassing FormMail’s attempts to validate a client-supplied `recipient` string, where most of these methods rely on various obscure and not widely known aspects of e-mail address parsing rules. These are by no means the only ways to circumvent these recipient address validation checks however. These checks may also be bypassed via at least three different exploits relating to FormMail’s use of regular expressions for e-mail address matching/validation. These additional exploit techniques are described below.

The first regular expression exploit we present offers an attacker an easy opportunity to get past FormMail’s recipient address checking tests, although perhaps in a way that will be quickly detected by an alert local administrator (of the exploited server). Note that due to the flow-of-control logic used within FormMail, an attacker *must* get past these tests at least once, or else no e-mail messages will be sent at all.

Consider the case where the local FormMail installer provided the following definition for the `@recipients` array:

```
@recipients = ("^john@our-server.tld");
```

Given that the string enclosed within the double quotation marks will in fact be used, verbatim, as a regular expression within a Perl regular expression match, and given that within such regular expressions, each period character actually stands for a single-character *wild card* character, an attacker could supply a value for the `recipient` CGI parameter such as:

```
john@our-serverXtld
```

In such a case, unless by some extremely remote chance, there happens to be another server on the same local network whose non-qualified node name is `our-serverXtld`, the mail message sent by the attacker through FormMail to the specified `recipient` address (and perhaps also to some additional set of recipient addresses smuggled in via the `email` and/or `realname` CGI parameters) will almost certainly generate a *undeliverable bounce message* for the non-

existent `john@our-serverXtld` address. Where will this bounce message go?

We would hope that the undeliverable bounce message would be directed to some responsible local administrator who would then quickly realize that the game is afoot, and that somebody somewhere is presently attempting to exploit the locally-installed copy of FormMail, but in point of fact this is unlikely to occur.

What is clear is the the bounce message, by convention and by SMTP standards, will be sent to the envelope sender address that was attached to the original e-mail message that generated the bounce. What will the original envelope sender address be?

Because FormMail itself generated the original message, and because FormMail itself was executed as a child process of the local web server, the original envelope sender address will typically be whatever local user-ID the local web server is run under, or rather, whatever local user-ID it changes to after it starts up.

For typical Apache installations, the account in question is, quite often, the local `nobody` account. That account name is, in turn, quite often and quite typically aliased (in the local mail server's aliases file) to `/dev/null`, in other words to the oblivion black hole.

The implications here should be apparent. An attacker who attempts to exploit FormMail may be easily able to escape immediately detection by local administrators simply by exploiting FormMail's inattention to the important detail that within the context of Perl regular expression matching, periods mean something other than literal periods. Exploitation of this inattention to detail may allow an attacker to "pass" FormMail's recipient address check in a very stealthy fashion. Once the attacker has gotten past the recipient check, at least once, then he may exploit the `email` and/or `realname` vulnerabilities of FormMail (described above) to also direct copies of his mail message to other targeted e-mail addresses.

Other, perhaps even more interesting variations on this theme are also possible, and are perhaps even more directly dangerous. A FormMail installer who had defined his `@recipients` array to:

```
@recipients = ("ourdomain.tld");
```

in order to allow FormMail to be employed to send mail to any user of the `ourdomain.tld` domain might later be unpleasantly surprised to begin receiving spam complaints from `somewhere.tld` users when some miscreant finally figures out that a recipient address such as:

```
user@somewhere.(ourdomain)tld
```

passes the (right-anchored only) regular expression match against `ourdomain.tld` with flying colors, only to subsequently have its parenthesized portion treated as a (non-significant) comment by the local RFC-conforming mail server.[14]

Yet another aspect of the weakness of FormMail's regular-expression based attempts to validate recipient address strings is its total insensitivity to important lexical aspects of domain names themselves, in particular, domain label boundaries.

FormMail installers who have elected to use one or more domain names in their definition of the `@recipients` array may be dismayed to learn that the lack of left-anchoring in FormMail's regular expression matching, together with FormMail's inattention to the importance and significance of domain label boundaries may lead to exploitation of a locally-installed FormMail, by outside spammers, to spam users in other domains whose own domain names have right-anchored substrings which just happen to exactly match the local domain name, even though there is otherwise no relationship between those other domains and the local domain.

For example, the following `@recipients` array definition:

```
@recipients = ("x.biz");
```

clearly allows the local FormMail script which contains it to direct FormMail-generated e-mail messages to various users having e-mail addresses directly within the `x.biz` domain. However what may be less clear is that it also allows FormMail-generated messages to be directed to persons having e-mail addresses within, for example, the `department-y.x.biz` domain. More troubling is that it also allows FormMail-generated messages to be directed to any person having an account in the (clearly unrelated) `generation-x.biz` domain. This problem arises because FormMail is not astute enough to understand that, for example, the domain name `generation.x.biz` is related to the `x.biz` domain while the domain name `generation-x.biz` is entirely unrelated. Thus, an instance of FormMail installed on a web server belonging to `x.biz` could potentially be used to spam users in the `generation-x.biz` domain.

Lastly, as we noted earlier, one method of circumventing FormMail's attempt to validate the recipient CGI parameter against elements of the installer-supplied `@recipients` array is to simply use alternative case for the user ID portion of the client-supplied recipient e-mail address. At the very least, this might allow an attacker to use FormMail to send an e-mail message to, for example, a user whose e-mail address is `JOHN@localhost.tld` even though the installer only intended to allow FormMail messages to go to the user whose e-mail address is `john@localhost.tld`.^[12]

This fact alone may be properly viewed as being essentially insignificant. At worst, the case insensitivity of the FormMail recipient address checks would seem to allow misdirection of FormMail messages to different (and unintended) users of the same target recipient system where some other valid and authorized FormMail message recipient also has an account.

Recall however that the logic of FormMail demands that at least one of the comma-separated substrings of the client-supplied recipient CGI parameter value must in fact 'pass' the recipient address validation check. Otherwise no e-mail message will be sent, regardless of whatever other clever manipulations an attacker might have undertaken.

Exploitation of the case-insensitivity of the `recipient` address checking code may provide yet another means for an attacker to pass the recipient address check, at least once, thus insuring that his desired e-mail message will in fact be sent by FormMail. Furthermore, it may allow the recipient address check to be bypassed even as it also allows the attacker to avoid immediate detection of his activities, either by any authorized FormMail message recipient or by any local administrator.

For example, let us suppose that the `@recipients` array was previously defined by the installer as follows:

```
@recipients = ("^john\@our-server.tld");
```

Based on the other vulnerabilities noted above, an attacker could easily spoof his way past the `HTTP_REFERER` check and he could then provide some clever values for either the `email` CGI parameter or the `realname` CGI parameter, or both. Those values could induce FormMail to send a desired e-mail message to some additional e-mail recipient addresses, above and beyond whatever address may have been provided by the attacker as the client-supplied `recipient` CGI parameter value. But even if he does all this, the attacker must still get past the script's checks on the `recipient` value, at least once, in order to induce FormMail to send any e-mail messages at all.

To get past the checks on the validity of the `recipient` CGI parameter, the attacker might set this parameter to the value `JOHN@our-server.tld`. This would cause the recipient check to pass, once, and that in turn would enable the script to continue further in its processing of the HTTP request, whereupon any of the several other FormMail vulnerabilities could then be exploited. In such a case however, the local mail server would also attempt to send a copy of the attacker's message to the e-mail address `JOHN@our-server.tld`. What then?

In the most common case, i.e. where `our-server.tld` is in fact the local domain, and where the mail server providing mail service for the local domain is either Sendmail or Postfix, the attacker will typically *not* be able to hide his activities via this sort of case-sensitivity exploitation because the local mail server (Sendmail or Postfix) will actually treat the user ID portion of a local e-mail address in a case-insensitive manner. Thus, even if the attacker tricks the FormMail script into attempting to send an e-mail message to `JOHN@our-server.tld` that message will still be delivered to the `john@our-server.tld` mailbox. The local `john` user will thus be alerted that the local FormMail script is being used, and possibly abused, and quick remedial action may then be taken to halt further abuse.

If however the specific mail server that provides mail service for the `our-server.tld` domain treats local user IDs in a case-sensitive manner, then the user ID `john` may exist, and may accept incoming e-mail, even though the user ID `JOHN` does not exist and does not accept e-mail.

In this case, any message sent to the `JOHN@our-server.tld` account will bounce as undeliverable. Where will it bounce to?

As noted previously, the message should be, and typically will be bounced back (along with an undeliverable notice) to its envelope sender address. And as was also previously noted, the envelope sender address will almost invariably be the local user ID under which both the local web server and any CGIs that it invokes run. That account, quite often, is the local `nobody` account, and e-mail for the local `nobody` account is, as often as not, aliased to `/dev/null`. Thus, if the attacker arranges to “pass” the FormMail recipient address validation test by changing the case of the user ID portion of some known *authorized* recipient e-mail address, and if the alternatively-cased recipient address is itself undeliverable, then this may represent an opportunity for the attacker. Specifically, it may allow the attacker to trick FormMail into believing that it has received at least one valid recipient address even though the message sent to that address will first be bounced as undeliverable and then, subsequently, will disappear, quietly and conveniently, down a black hole, *without* alerting any local end-users or any local administrators on the exploited system that exploitation is occurring.

It should additionally be noted that even in cases where the web server itself is run under some account other than the `nobody` account, the account that the web server is in fact run under will often and typically be one for which incoming e-mail messages, including bounces, are dumped into a black hole. Furthermore, even for web server execution accounts that are not aliased to `/dev/null`, local administrators may not be expecting any incoming e-mail for the web server execution account, and thus, any e-mail that *does* arrive for that account may be monitored only rarely or perhaps never. Such a situation is exactly what an attacker wishing to escape immediate detection would desire.

In summary, it may safely be said that the case insensitivity of the FormMail recipient address checking code may open up additional possibilities for quietly covering up abuse and exploitation of the other FormMail security flaws detailed herein.[13]

Mail-Bombing by Proxy Using FormMail

Although exploitation of installed FormMail scripts for spamming purposes is far and away the most likely scenario involving misuse and abuse of FormMail, a desire for completeness compels us to note some additional ways in which typical FormMail installations may be abused to cause harm or annoyance to others, even if only indirectly.

In general, any Internet miscreant may easily flood any Internet e-mail account of his choosing by simply sending a flood of e-mail messages directly to that target account. This direct approach is however known to be a good way to have one’s outbound e-mailing privileges revoked in short order.

Being aware of this fact, many Internet miscreants bent on performing an act of *mail bombing* against a given target e-mail account will instead elect to do so only indirectly, employing some intermediary agent or system that can be induced to send e-mail messages to arbitrary attacker-specified e-mail addresses. To the extent that the intermediary prevents the final victim of the mail bombing from easily learning the identity and/or IP address of the mail bomb originator it

will be seen as a potentially useful tool for a *stealth* mail bomb attack.[15]

Given these facts, together with the multiple FormMail vulnerabilities described above, it should be immediately seen that any installed instance of the FormMail script may be employed, in various ways, as part of a *stealth* mail bombing attack directed against an arbitrarily selected target e-mail address.

It has certainly been shown above that FormMail may be induced to send arbitrary messages to arbitrary target addresses in a way which prevents the final recipient from being able to easily determine the IP address of the actual message originator. This fact alone could provide the basis for a stealth mail bombing attack. Additional possibilities for such attacks, involving FormMail, may also exist however.

For example, if FormMail is being employed on a given web site in conjunction with an e-mail *auto-responder* of some kind, then it may be possible for an attacker to repeatedly invoke FormMail while setting the email CGI parameter to the e-mail address of his intended victim. The victim in this case will receive a flood of “auto-responses” for messages that he himself did not originate, and these auto-responses may (and quite often will) contain no indication of the IP address that originated whatever message or messages have triggered the auto-responder to send a response message. Note also that even for a relatively “smart” auto-responder that makes it a point to attach the e-mail headers of each original *triggering* e-mail message to each of the auto-response messages it generates, the trace information provided by such headers will lead back only as far as the web server on the system that hosts the FormMail script. That information will of course be insufficient to determine the IP address used by the actual instigator of the mail bomb.

Because of the above scenario, it is strongly recommended that FormMail never be intentionally employed in conjunction with any kind of e-mail auto-responder.

[1] The canonical reference for the original ‘FormMail spamming’ advisory is:

<http://www.securityfocus.com/archive/1/168177>

Note that even earlier, reports had surfaced regarding FormMail and environment variable information leakage:

<http://packetstorm.decepticons.org/advisories/blackwatchlabs/BWL-00-06.txt>
<http://www.securityfocus.com/bid/1187>

and a fix was developed for this problem:

<http://www.securityfocus.com/archive/1/62033>

A remote command execution vulnerability was also reported (Aug, 1995) for the 1.0 version of FormMail:

<http://www.securityfocus.com/bid/2079>

[2] The original report of possible widespread scanning of the net for vulnerable FormMail scripts may be found at:

<http://www.extremetech.com/article/0,3396,s%253D25124%2526a%253D18236,00.asp#story4>

[3] The home page for one allegedly “fixed” version of FormMail may be found at:

<http://www.mailvalley.com/formmail/>

An entirely separate “fixed” version of FormMail may be found at:

<http://nms-cgi.sourceforge.net>

Unfortunately, neither of these security-enhanced versions of FormMail address all of the security issues raised herein.

[4] Inexplicably, even though versions of FormMail prior to 1.9 are known to be exploitable (as what amounts to anonymizing open mail relays) the author of this script continues to distribute several of his earlier, insecure versions of the script (for use with/on various versions of MS Windows) via his web site and the FormMail home page. This will naturally tend to contribute to a worsening of the existing global problem of insecure FormMail installations, over time.

[5] We use the term *anonymized* herein to indicate that the IP address of the actual message origination point will not be present in any of the e-mail `Received:` headers contained in the message themselves. This lack of tracing information makes it essentially impossible for a message recipient to determine the actual origination point, at least not without the aid of the administrator of the server that hosts the FormMail script and not without a careful search of log files by that administrator.

[6] The lynx web browser provides the `-noreferer` command line option to achieve this exact effect.

[7] Javascript, in particular, implements a `submit()` built-in primitive that can be used to simplify attacks on web CGIs. The authors have found no way to induce Javascript’s `submit()` primitive to suppress the inclusion of a `Referer:` header in the HTTP request generated by a call to `submit()` however.

[8] There are, of course, other ways to cause typical web servers to ignore some rightward part of a complete URL. The forward slash character (`/`) can also be used to achieve the same effect when the portion of the URL to the right of the relevant forward slash can be located by the

relevant web servers and when it is an ordinary file (as opposed to a directory).

[9] There are, of course, other ways by which one could arrange for `Referer:` HTTP headers to be suppressed, the most obvious of which is to make use of any of the Internet's many *anonymizing HTTP proxy servers*.

[10] One of the authors of this advisory (Guilmette) is currently maintaining a list of the IP addresses of such sites for spam blocking purposes.

[11] Please see:

`http://web.nps.navy.mil/~miller/percent-hack.html`

for a brief description of the traditional (if non-standard) *percent hack* e-mail addressing notation.

[12] Of course, different operating systems have different conventions and practices with regards to the case sensitivity... or lack thereof... of local user IDs. It should be noted however that FormMail is most frequently installed on UNIX or UNIX-like systems, and that these systems do, in virtually all cases, treat local user IDs as being case sensitive, thus allowing for the possibility that there might be a user whose ID is `john` and also another user whose ID is `JOHN`, both on the same single system.

[13] Webmasters may also wish to reconsider the advisability of running web servers under accounts whose incoming e-mail is either sent to `/dev/null` or ignored. It may actually be wiser to select some account other than the `nobody` account to run a web server under, and then to alias that other account to `root` or some other frequently-monitored local administrator address. The `nobody` account should more probably be preserved as an effective e-mail alias for `/dev/null` however, as there are certainly instances in which you really don't want any bounces, no matter what.

[14] See RFC 2822, especially its discussion of parenthesized comments and e-mail address syntax.

[15] An indirect mail bomb attack against a given e-mail address may be easily undertaken simply by sending a large number of e-mail messages to some non-existent and/or otherwise undeliverable e-mail address on some specific third-party *intermediary* system with the envelope sender (i.e. bounce back) e-mail address set to the e-mail address of the intended victim. Such an indirect *mail bombing via bounces* attack is not at all stealthy however, because in the vast majority of cases the attacker's IP address will be shown clearly in the `Received:` headers of the messages that triggered the bounces, and these in turn will typically be included in the bounce messages received by the final victim.