

Using PHP to Automate Security



Table of Contents

Online at: <URL:<http://Ken.Coar.Org/slides/phpblock/>>.

PHP can be more than just a scripting language used for constructing dynamic Web pages at request time. One application is to use its integration with the Web server to record — and potentially deal with — security issues.

- [What's the Point?](#)
- [Why PHP?](#)
- [How to Deal with Malbots](#)
- [Problems and Pitfalls](#)
- [Tracking Requests](#)
- [Denying Access](#)
- [Integration with mod_security](#)
- [Defining Misbehaviour](#)
- [Spider Traps](#)
- [Dynamic robots.txt](#)
- [To Be Done](#)
- [References](#)

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 1 of 13.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

What's the Point?



There are species of malware which behave in unfriendly fashions toward Web sites. In particular, spiders and Web crawlers which hammer them with an excessive number of requests, or which duplicate content elsewhere without permission, or harvest email addresses, or wander into forbidden areas, or spam Web logs, or spam the referrer log, or ...

Between [mod_security](#) and `mod_rewrite`, among other methods, you can put together a suite of scripts that can track undesirable requests aimed at your server — and deny access to those you decide are deliberate.

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 2 of 13.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

Why PHP?



The main reason for using PHP is to allow things to happen in real time. For the most part, the Apache `httpd` server relies on configuration information either read once (the system configuration files), or on each request to a specific directory tree (`.htaccess` files). Unless you update the `.htaccess` file each time a determination/decision is made concerning a requesting client, this doesn't provide for real-time responsiveness. And it doesn't address the issue of documents that aren't in the filesystem, either.

So this specific method actually uses active PHP scripts in place of normally static files. For instance, the `robots.txt` file is actually a PHP script, which can make notes in the database about clients requesting it and tailor its output specifically to the client involved.

The dynamic/learning blocking aspect is provided by a `RewriteMap` programme, which is consulted on each request. This programme is also written in PHP, but it's a standalone script rather than a Web page.

This method doesn't necessarily scale well to large sites, and it requires both the PHP standalone interpreter and the Apache `mod_php` module. However, it at least illustrates the possibilities.

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 3 of 13.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

How to Deal with Malbots



In order for a Web server to be able to deal with malbots, it needs to be able to recognise them. There are at least three main ways of doing this:

1. **By source IP address.** Some malbots are run by specific companies from their own servers. If you know the IP addresses of these servers, you can use that information to deny them access.
2. **By user-agent.** Most clients, including many malbots, include information about the software they're using in each request they make. This is called the 'user agent' identification, and this can be used to block malbots that always use a known string.
3. **By referring page.** When you follow a link from one page to another, the first is called the *referrer*. (In the network specifications, this is incorrectly spelt as 'referrer'. O well.) Some malbots always specify particular referrers, which fact can be used to identify them.
4. **By [mis]behaviour.** For example, a malbot which scans your site in defiance of your `robots.txt` settings, or which checks `robots.txt` and then ignores it, is clearly behaving badly. Other clues include SQL statements or JavaScript embedded in the URL; referring URLs containing key words; and spam aimed at wiki pages or Web log comments.

Copyright © 2004 by Ken A L Coar. All rights reserved.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

Problems and Pitfalls



Unfortunately, the first three detection methods (source IP address, `User-Agent`, and referring page) are far from foolproof. In our current environment, with DSL, cable, and other high-speed connexion types becoming more and more common, assigning a new DHCP-supplied IP address to customers each time their systems come online, there's no assurance that the IP address used to probe your system today won't be assigned to a totally innocent individual tomorrow. User agents exhibiting bad behaviour might be doing it systematically, such as an address scraper — or incidentally, such as if an actual person enters a forbidden URL out of curiosity. And not only is the `Referer` request header field easily spoofed (hence 'referrer spam'), but a lot of people systematically *don't* send it in order to keep their browsing patterns private.

As time passes and you review the tracking records, you'll probably be able to identify combinations that are indisputably perps — or perhaps you'll collect such information from friends or other sources. It's a pretty sure bet, however, that you're going to need to do a fair amount of manual inspection of the records.

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 5 of 13.

This slide last modified at Friday, 01 April 2005 10:40:18 EST

Tracking Requests



One of the keys to this whole detection and prevention task is tracking questionable — or not-so-questionable — requests made to your server. Let's set up a fake URI to which we can rewrite problematic requests:

```
RewriteRule "^/coroner" "/admin/record-request.php" [PT]
```

There will be multiple circumstances under which we'll redirect things here; we can include more information through envariables or the query string:

```
#  
# Pass reason for invocation through the query string  
#  
RewriteRule "viagra" "/coroner?reason=URL-keyword" [PT]  
#  
# Pass it through an environment variable  
#q  
RewriteRule "offensive" "/coroner" [PT,ENV=REASON:URL-keyword]
```

Bear in mind that, by default, `Rewrite*` settings aren't inherited from the main/global scope. Therefore, you may need to add a line such as

`RewriteOptions inherit`

to each of your `<VirtualHost>` sections. And beware that there aren't rules in the global scope that you **don't** want inherited!

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 6 of 13.

This slide last modified at Friday, 01 April 2005 10:40:18 EST

Denying Access



The tracking and detection portion should now be pretty much a SMOP. The part that makes responsiveness dynamic, however, lies in how you determine that a particular request should be rejected.

Apache is big on static configuration files, but not quite so flexible when it comes to changing its operation at run-time. Fortunately, `mod_rewrite` has the ability to base some of its decisions on external data sources — which **can** be modified at run-time, and thus affect the server's operation. Specifically, there's the `RewriteMap` functionality:

```
RewriteMap denybyip txt:/usr/local/apache/htdocs/admin/malbots-by-ipa
RewriteMap denybyua txt:/usr/local/apache/htdocs/admin/malbots-by-ua
RewriteMap denybyrf txt:/usr/local/apache/htdocs/admin/malbots-by-referrer
RewriteCond ${denybyip:%{REMOTE_ADDR}|NOT-FOUND}      !=NOT-FOUND [OR]
RewriteCond ${denybyua:%{HTTP_USER_AGENT}|NOT-FOUND}   !=NOT-FOUND [OR]
RewriteCond ${denybyrf:%{HTTP_REFERER}|NOT-FOUND}     !=NOT-FOUND
RewriteRule .* - [F,L]
```

These directives will return a 'Forbidden' status to any client whose IP address is listed in one or more of the `malbots-by-ipa`, `malbots-by-ua`, or `malbots-by-referrer` files. The above implementation defines those as simple text files, but a more

comprehensive (or larger) installation might well replace those with some sort of database back end.

This allows the `record-request.php` script to update either or both of these files according to whatever rules you like.

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 7 of 13.

This slide last modified at Friday, 01 April 2005 10:40:18 EST

Integration with mod_security



mod_security is a handy tool specifically designed to help protect your Web server from 'standard' attacks. With a little bit of work, you can tie it right into the PHP interdiction suite and move on to more complex issues.

```
<IfModule mod_security.c>
  SecFilterEngine On
  SecFilterDefaultAction "deny,log,status:505"

  SecFilterCheckURLEncoding On
  SecFilter "/_vti"
  SecFilter "\.exe"
  SecFilter "/[Ff]orm[Mm]ail\."

  ErrorDocument 505 /coroner/interdict?reason=mod_security
</IfModule>
```

Here we tell *mod_security* to respond to requests that match its criteria by raising a 505 status. HTTP code 505 is defined as 'version not supported,' and we're using it for two reasons:

1. The Apache Web server will only let you use 'known' status codes in `ErrorDocument` directives, so we have to pick one that's already defined, and

2. It's extremely unlikely that this status code will ever be *genuinely* used except under exceptional circumstances.

In the same block, we include an `ErrorDocument` directive which will cause any 505 status conditions to be handled by our tracking script.

In fact, we're adding `/interdict` to the handling URI, and we can have the script recognise that as meaning 'definitely disallow this, don't just record it.'

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 8 of 13.

This slide last modified at Friday, 01 April 2005 10:40:18 EST

Defining Misbehaviour



Recognising a perp by IP address, user agent, and/or referring page URL is quite straightforward. More complex, though, is recognising misbehaviour. What constitutes misbehaviour is extremely site-specific; the types covered here are abuse of the `robots.txt` mechanism and the Bluebeard syndrome (going where you've been explicitly told not to go).

Here are the specific behaviours considered in this presentation:

- Falls into a spider trap -- follows a link not visible to users
- Disregard of `robots.txt` -- accesses an area denied to robots
- Lies about obeying `robots.txt` -- requests it but ignores prohibitions
- Using `robots.txt` to *look* for prohibited content

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 9 of 13.

This slide last modified at Friday, 01 April 2005 10:40:18 EST

Spider Traps



A *spider trap* is a link that ordinarily won't be followed by a human being, and which points to a URL that will result in the follower being recorded. To keep people from following it, it's made as invisible as possible, and clearly labelled with 'do not click here!' warnings. In some ways this is similar to a 'Web bug,' save that it is entirely passive.

The problem with spider traps is that they need to be embedded in actual content, which means they require the cooperation of whomever owns the content.

To simplify things, I just add a line such as this in my content:

```
<?php Include($_SERVER['DOCUMENT_ROOT'] . '/scripts/spidertrap.php'); ?>
```

Including a spider trap

And here's the relevant code from the `spidertrap.php` script:

```
$warning = 'Do not follow this link, or your host will be blocked '
```

```
. 'from this site. This is a spider trap.';

print "<a href=\"/coroner/interdict?reason=spidertrap\"\n"
. " onclick=\"return false;\"\n"
. " onmouseover=\"window.status='$warning'; "
. "return true;\"\n"
. ">"
. "<img height=\"1\" width=\"1\" align=\"right\" border=\"0\"\n"
. " src=\"/images/transparent_1x1.gif\"\n"
. " alt=\"$warning\" /></a>\n";
```

The `parlour.html` document is another script which will record information about the client for possible inclusion in a interdiction list.

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 10 of 13.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

Dynamic robots.txt



The [robot exclusion standard](#) was created as a way for scanning robots to *politely* ask what parts of a site they should process, and which they should ignore. Originally developed as a hack solution to a new problem, it has nevertheless become a *de facto* standard, and well-behaved robots check for it and obey it if found.

There are a number of different types of nasty robots out there. One of them is, of course, the type that completely ignores the `robots.txt` file. Another more insidious type reads the `robots.txt` file, but ignores it. Even worse are those that read the file and **explicitly** seek out the forbidden areas.

The first type can often be identified by falling into [spider traps](#). The third type can also be caught in spider traps by listing the traps' URLs in the prohibition section of the `robots.txt` file. It's the second type — the one that reads `robots.txt` but ignores it — that's the most difficult of the three to catch.

To identify these, you need to know two things:

1. That the robot requested the `robots.txt` file; and
2. that it subsequently requested something the file forbade,

You might want to give different robots access to different location; for instance, you might want to let Google look in some places you don't want an address scanner to

index. The `robots.txt` format allows you to list *per-client* restrictions — but an unscrupulous robot *could* use an all-inclusive `robots.txt` file to find out what other user-agents can see, so it can masquerade as them. Having the `robots.txt` file be dynamically generated allows you to deal with this. It also makes it easy to record which clients actually *request* the file.

My `robots.txt` file is a PHP script, and it produces output according to the client and the rules in a text file. Here's a sample rules file:

```
#
# Paths on the server to be forbidden through the robots.txt
# mechanism.
#
# Format:
#   user-agent:path
#
# If the user-agent is '*' it will always be listed.  Otherwise
# it's treated as a regex.
#
#
# Don't let the scripts be indexed.
#
* : /cgi-bin/
#
# Other private places.
#
* : /private/
```

```
* : /archives/  
* : /~  
#  
# Here's a 'honeypot' -- this location doesn't actually exist anywhere,  
# nor is it referenced any place except here and the config file, so  
# if anyone ever accesses it they must have read this file and are  
# explicitly trying to access forbidden places.  Bad bad bad.  
#  
* : /user-email-addresses/  
#  
# And here's the pit to which the spider traps lead.  Nothing should follow  
# them there..  unless it deserves to die.  
#  
* : /coroner/interdict  
  
#  
# These are the valid ones we let crawl according to the above restrictions.  
# A blank path means 'don't list this one individually in robots.txt output,  
# it has no specific restrictions.'  
#  
Blogbot :  
daypopbot :  
FAST-WebCrawler :  
www.galaxy.com/info/crawler.html :  
Googlebot :  
Gulper :  
scooter :  
timboBot :  
  
FindGifs : /gallery  
#  
# Add valid per-robot restrictions above this point; below is the
```

```
# catch-all for those which don't have their own.  If we don't
# know who they are, restrict everything by default.
#
.* : /
```

Here's a sample of the `robots.txt` document generated for an approved agent:

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /private/
Disallow: /archives/
Disallow: /~
Disallow: /user-email-addresses/
Disallow: /coroner/interdict
```

Here's the `robots.txt` output generated for a semi-restricted known spider:

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /private/
Disallow: /archives/
Disallow: /~
Disallow: /user-email-addresses/
Disallow: /coroner/interdict

User-agent: FindGifs
Disallow: /gallery
```

Find finally, here's the `robots.txt` output generated for any other client (we assume it's a malbot):

```
User-agent: *  
Disallow: /cgi-bin/  
Disallow: /private/  
Disallow: /archives/  
Disallow: /~  
Disallow: /user-email-addresses/  
Disallow: /coroner/interdict
```

```
User-agent: EmailCollector  
Disallow: /
```

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 11 of 13.

This slide last modified at Friday, 01 April 2005 10:40:59 EST

To Be Done



-
1. Associate the user-agent/referrer/IPA of a particular request when they're stored, so it's clear why each entry is in the database.
 2. Publish sample scripts.
-

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 12 of 13.

This slide last modified at Monday, 28 March 2005 14:08:36 EST

References



-
- Pilgrim, Mark, [How to block spambots, ban spybots, and tell unwanted robots to go to hell](http://diveintomark.org/archives/2003/02/26/how_to_block_spambots_ban_spybots_and_tell_unwanted_robots_to_go_to_hell), 2003. http://diveintomark.org/archives/2003/02/26/how_to_block_spambots_ban_spybots_and_tell_unwanted_robots_to_go_to_hell
 - Gunton, Neil, [Stopping Spambots: A Spambot Trap](http://www.neilgunton.com/spambot_trap/), 2004. http://www.neilgunton.com/spambot_trap/

[Ken A L Coar](#)

Copyright © 2004 by Ken A L Coar. All rights reserved.

Slide 13 of 13.

This slide last modified at Monday, 28 March 2005 14:08:36 EST