COMPSCI334S1T2008 Assignment 4

Lecturer: Ulrich Speidel, ulrich@cs.auckland.ac.nz

Due: May 28, 2008, no later than 23:59 pm via the assignment dropbox

PHP Security

PHP is a pretty secure language for dynamic web programming. Conventional languages (such as C) used to provide a number of traps for unwary programmers. PHP circumvents a few of these. This applies in particular to the fact that in PHP, you don't have to allocate memory and hence there is practically no chance of creating an exploitable buffer overflow.

Nevertheless, there are still quite a number of issues that you need to consider when you are programming. A few of these will be demonstrated in the lectures. The purpose of this assignment is to sharpen your eye for such security-related problems, especially those specific to scripts that take user input, interact with a database, and output data from third parties to a user's browser.

In this assignment, you don't have to write a program – your task is to find the security holes in an existing script and fix them.

You'll have to set up a database (more instructions on this below) and a little PHP application. The script sources and the database table setup file are both available from the course web page.

To give you a head start, there is a little article called **`safe_php.html`** in the ZIP file. It contains a description of the pitfalls and the kind of security holes you need to watch out for. Read this file first before you even look at the application.

The application

The script itself is a simple message board application. I'll describe what it does in an ideal world (i.e., where nobody takes advantage of security holes). You'll find that the security holes in the script make a mockery of much of the following functional description. However, once you're finished with your assignment, these holes should be closed and the description should apply.

Here we go:

The message board has an *admin* user. The admin is the only user that can add and delete other (non-admin) users. The admin is also the only user who can delete other users' messages from the system. When a user is deleted,

all of his/her messages (which are stored in a file with the username in the *messages* subdirectory) are deleted, too.

Beyond the message board functionality, the admin is not able to modify the site in any way, i.e., the admin should not be able to access the file system or issue other commands on the server. The admin should also be able to change his or her username and password, which he/she needs to log in.

Ordinary users also need to log in with a username and password, which they can also both change to a name/password they like (unless the new username is already in use). They can post a message to the board, but they cannot delete any.

So far, so good. Here is some more detail:

If the main script, **`index.php`**, is called without any POST variables, it simply displays a page asking for a username password. These usernames and passwords are contained in a MySQL database which you will need to install (see instructions below).

The SQL script used for this purpose has a dummy entry for the admin password, which you should change this before you install the database. You will also need to configure the database access information in the main script. The database access password in **`index.php`** should not be the same as the admin password, but it should be changed before you install the script. As you may get locked out of the database if you get hacked, there is also a script in my area of the server that lets you reset your database password (see below).

When you log in successfully, the application shows a list of messages that are already on the system. This list is initially empty. The script then also shows two buttons at the bottom of the page. These take you to forms that let you submit a message and that let you change your username and password, respectively.

If you are logged in as admin, you also see two buttons next to each message, which let you delete the message, or even the user that sent the message (and simultaneously all messages that this user sent).

All forms are included by the main script as required using the **`include()`** function in PHP from files whose main filename is written in capital letters, e.g., **`NEWPASS.php`**.

All buttons are surrounded by hidden forms (which you can see with View…Page Source in your browser), with the respective button being the only visible element. Once a user is authenticated with the script, the username and password are included in hidden fields in these forms. When one of the forms is submitted, the hidden fields are also sent to the server to permit re-authentication. A hidden field with name “MODE” determines the course of action for the script to take. E.g., if MODE has the value

"SAVEUSER", the script will attempt to save the submitted user data as a new user.

Getting started

Your first assignment task is to read the safe PHP scripting guide. You can find it on the assignment page of the course home page (it's also in the zip file as **`safe_php.html`**). I expect you to have read and understood the guide as part of your assignment - all its contents are examinable in the final exam! All security holes relevant to his assignment are described in the guide.

As background, you should then also read the PHP documentation on the configuration directives **`magic_quotes_gpc`**, **`magic_quotes_runtime`** and **`register_globals`**.

Carrying on

Now read the code of the assignment, i.e., read and try to understand **`index.php`** and the files it tries to include. Now that you have read the guide, **identify the security holes in `index.php` and patch them - before you try out your assignment on web334.** None of the other files should be used to patch problems – they will be discarded by us before marking. How do you find the security problems? **Read the guide!**

The next task is to ensure that your script works with arbitrary settings of the three configuration directives. That is, you need to define a coding policy for yourself that determines how user data is accessed by the script (you'll find that this is already done for you by always using **`$_POST`**) and what its backslash escape status will be when the data is processed, saved into the database, retrieved from the database for further processing, and when it's being output to the screen. Screen output that shows backslashes will lead to point deductions! Note that knowing the backslash escape status of your data goes hand in hand with fixing security problems.

Is this a security hole?

In order to define the scope of what constitutes a security hole, I want you to assume that:

- third-party spying on the HTTP or the database connection is not an issue here (e.g., because the final server that the script gets placed on runs SSL) and that pages are viewed only from secure locations (e.g., own machine, not shared machine). In other words: It's OK to store the password in plain text in an HTML page returned to someone who has already authenticated himself/herself with that password. In practice this wouldn't be that great, but for this assignment we presume it's OK.
- no further security measures have been taken on the server side

- anything that is remarked-upon in the file as “this is not a security hole in its own right” isn’t a security hole. E.g., storing the DB password in plain text in the file isn’t a security hole, unless another hole permits an attacker to actually read the contents of the file.
- some security holes may only be exploitable under certain circumstances that depend on the settings for **`register_globals`**, **`magic_quotes_gpc`**, and **`magic_quotes_runtime`**. As mentioned above, you can find information on these in the PHP documentation. It is up to you to find out the settings on web334 at any given time (they will change)
- the absence of an index file in the messages directory is not regarded as a security hole.
- the fact that users can post an arbitrary number of messages is also not a security hole.

Things that qualify as security holes:

- Any flaw in the script that lets anyone log in without knowing the password of a registered user
- Any flaw in the script that lets anyone or a normal registered (non-admin) user log in as admin
- Any flaw in the script that lets anyone, including the admin, modify the database in an unintended way
- Any flaw that gives access to another user’s password or account
- Any flaw that gives anyone the opportunity to read or write files that they should not be able to see, create, modify, or delete
- Any flaw that lets anyone specify PHP or shell code to be executed on the server
- Any flaw that lets anyone corrupt HTML code displayed to another user, potentially leaving their view of the site dysfunctional (e.g., by smuggling in a tag that prematurely terminates a form). This includes the injection of HTML or JavaScript code that triggers HTTP requests or reveals information such as passwords to the attacker (XSS attacks)
- Any flaw that lets an admin create additional users with admin privileges.
- Any flaw that might permit SQL injection. Note that the mysql client library doesn’t permit more than one query to be executed via **`mysql_query()`** at a time. However, that could change, so SQL injection attacks that convert a single query into multiple queries must be prevented.

You may wish to create custom HTML forms or PHP scripts to help you attack your own script (or those of other students).

Setting things up on the web server and the database server

Download **`messageboard.zip`** from the assignment page on the course website. It contains a number files all of which are part of the site (including

the **`messages`** subdirectory), with the exception of **`messageboard.sql`**, and **`safe_php.html`**.

Setting up the database (do this first)

In order to set up the database, ***first*** go to the URL:

https://web334.cs.auckland.ac.nz/db/changedbpw.php

You will be asked to enter your university login, before being taken to the password changer page. On this page, change your database password to **something that is not your NetLogin password or your student ID**. You can change the password as often as you like (e.g., after your script has been hacked by a classmate).

After you have changed your database password, transfer the file **`messageboard.sql`** to **`web334.cs.auckland.ac.nz`** using sftp2 (or sftp, or Putty, or whatever you used to transfer **`booking.php`** in the second assignment). Put it into your home directory (the directory above **`public_html/`**).

Now use ssh (or ssh2, or whatever) to log into web334, using your NetLogin username and password. At the command prompt, enter:

`/usr/bin/mysql -u <your-upi> -p <your-upi> < messageboard.sql`

This will prompt you for your database password. After you have entered your password, your database will be set up. Note that the database name is your UPI and the database password is whatever you set it to via the password changer page. You can use the same procedure to restore the database if it gets overwritten by a fellow student hacking into your script.

Setting up the website itself

In order to put your assignment onto the web and make it visible, you must complete the steps described in Assignment 2, except that this assignment goes into a folder called **`ass4/`** inside **`public_html/`**. Give the web server lookup, read, write, delete, and insert permissions for the content of the “ass4” directory:

```
chmod –R 775 ass4/
```

This setting lets your classmates exploit some of the security holes in the script. If you think you did a good job fixing the holes, you shouldn’t have to be afraid of these settings. If you didn’t do a good job, chances are you’ll find out the hard way ☺.

Then create another subdirectory named "images" inside the ass4/ directory.

Now put a (fixed) copy of **`index.php`** and the other scripts into this **`public_html/ass4/`** directory. Make sure you keep a backup of the files somewhere offline – you may lose these files to a hacker if your assignment is not watertight and your classmates are smart!

The script should then be visible as

http://web334.cs.auckland.ac.nz/~<your upi>/ass4/index.php

See the note on hacking at the end of the assignment sheet!

Submission

Please submit via the assignment dropbox:

https://adb.ec.auckland.ac.nz/adb/

The only file that will be marked is **`index.php.`**

DO NOT USE A DIFFERENT FILENAME – OR YOUR ASSIGNMENT WILL NOT BE MARKED.

Do's and Don'ts

Naming conventions for files, database, and form fields

You must not change the names of any of the script files or that of the database. The database definition must not change either, except for the admin password in the user table. Similarly, **the names of the form fields in your submission must not differ** from the ones used in the original **`index.php`** script and its associated forms, and you must not add any fields. Files that do not meet these criteria will not be marked.

Copying

You must not copy other people's code. The above assignment contains enough scope to exclude the possibility of similarities between individual solutions. If your code looks similar to that of other people, then you will get zero marks. We will not accept any appeals. Do not ask other students for their code. Do not offer them yours. This includes not letting other students see the contents of your folders.

Hacking into other people's scripts - please note!

It is the nature of this assignment that your script and that of other students may be susceptible to attack via HTTP. As a learning exercise, trying to hack your fellow student's scripts through the security holes in the scripts is

explicitly permitted and encouraged for this assignment (and for this assignment only).

You are allowed proof-of-concept hacking only and you must immediately inform the student concerned by e-mail that you hacked his or her site. Any damage you do must be restricted to the ass4 directory or its subdirectories, and/or to the victim's database. Proof-of-concept hacking nevertheless means that you are permitted to delete files in the student's ass4 directory or its subdirectories, if you can – it is everyone's responsibility to keep a safe backup. You must not use the security holes in order to upload objectionable content, or deface the victim's web site in order to insult or ridicule them (a simple note that this web site has been hacked by you will do). Note that we are logging HTTP requests to web334 and will be able to trace you.

This means that it's a really good idea to keep a backup of your work somewhere away from your web directory, and to ensure that you only upload a "tight" script. In the last two years, a number of students lost work because they did not keep a backup!

One last note: Don't brag on the forum if you hacked a classmate. Only script kiddies do that.