Lab #5

Daniel Graham

SDEV 300 7981

Building Secure Web Applications – Dr. Jason Cohen

# Manual interception techniques and analysis

Manual techniques are going through the application with the ZAP proxy running, looking at source code.

First thing I did was start the ZAP application and set Firefox to run through the proxy. Logging into the app manually with a random userid and password works.

Entering the Path “C:\Bitnami\wampstack-7.1.16-0\apache2\htdocs\SDEV300\week5\Lab5Mod\data\secrets.txt” for the filename in the app exposes the information on the deleteme.php file.

This is a massive security flaw as it allows a remote user to access the filesystem of the server and read **ANY** file.

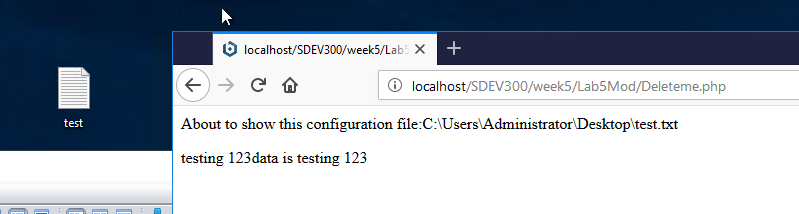


Figure 1 - accessing any files on the system

Figure 1 as you see above is a file called test.txt on the desktop, the web application was able to access the file and its contents by simply knowing where to look. This means that an attacker could also use absolute references to target a specific file. Such as ../config/database-credentials.php

For example, see Figure 2, which shows where I inserted in the SIMPLE text of “saveme.php” since I know the file name, now I can see the source code of the file! This can also enable Cross site scripting attacks by the user being able to enter whatever they want into the textarea due to the next page just echoing out $file. As you can see in figures 3, and 4Having a file reading function accessible to the public is a VERY bad idea.

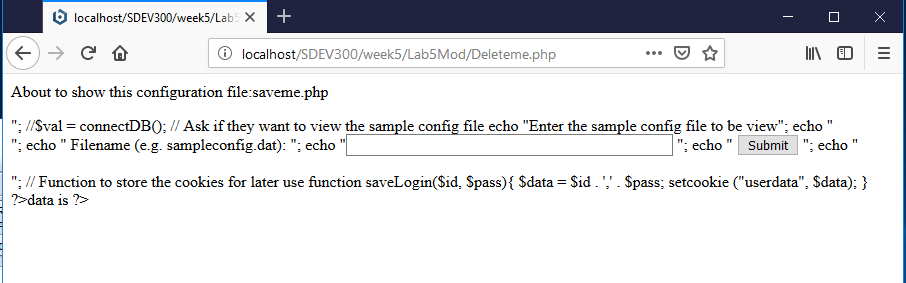


Figure 2 - Viewing a php file's source code from app

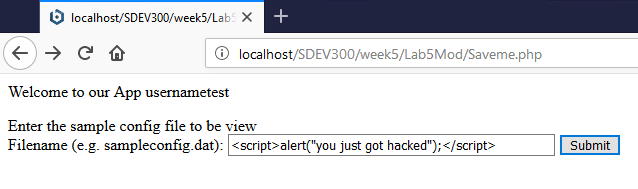


Figure 3 - inserting js into the form

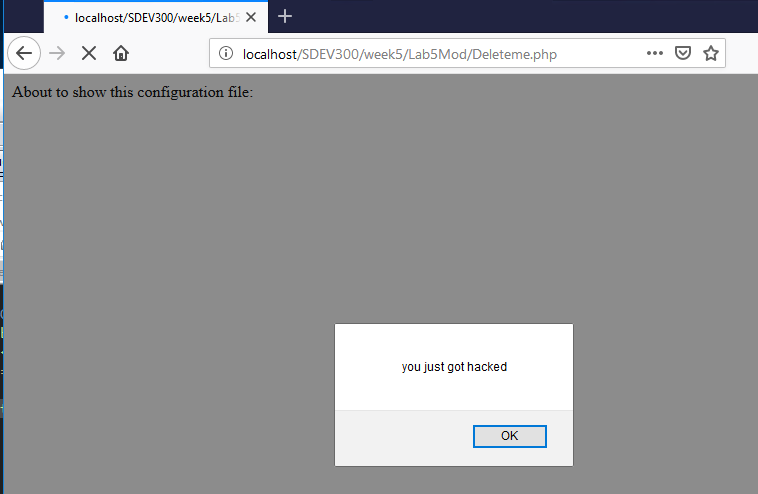


Figure 4 - JS running on deleteme.php

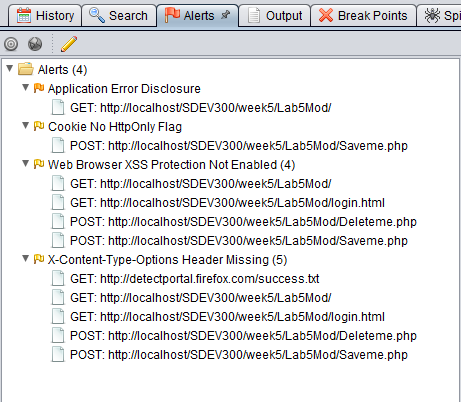


Figure 5- alerts while browsing manually

Next, through manually going through the website while ZAP was running, I see a few problems as shown in figure 5.

First there is an error because the directory is exposed at /Lab5Mod/ this is fixed by adding .htacess file and adding the following:

Options -Indexes

The Second on the list is “Cookie No HttpOnly Flag” this is because the cookie does not have the httponly flag set when it was created. This is important because it helps prevent a client-side script from accessing the cookie and its data.

The third alert is XSS protection not enabled. This is due to a web server configuration problem. Adding the following to the .htaccess file will resolve the problem:

<IfModule mod\_headers.c>

Header set X-XSS-Protection "1; mode=block"

</IfModule>

The next issue is the content-type-options header missing which is just setting content type to nosniff:

<IfModule mod\_headers.c>

Header set X-XSS-Protection "1; mode=block"

Header set X-Content-Type-Options nosniff

</IfModule>

Next by manually analyzing the code of saveme.php the user-defined function saveLogin is being called to save the login (especially the password) to a cookie, this is NOT secure as cookies are hijackable and human readable. The solution is to move to sessions. I added a session to page, along with insuring the user is logged in to access deleteme.php and saveme.php, and if not the user is redirected back to the login.

# Automatic Detection

Figure 6 shows the automatic scan ran on <http://localhost/SDEV300/week5/Lab5Mod/>

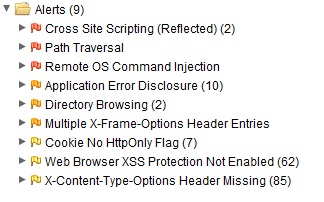


Figure 6 - Alerts for Lab5 Modified

I will walk through these from the most critical (the red flags) to the least (yellow flag). Cross Site Scripting is an extremely important alert as it can and probably will compromise your users. These two Alerts were on DeleteMe.php and Saveme.php. The changes that need to be made in both files is escaping the passed information. In a real world scenario, the username or password would not be valid, and it would be needed to validated through a registration process, the login process would simple validate the credentials, and not insert or display on a page directly.

The next Alert that the automatic detection noticed is path traversal. This is something that noticed via manual code inspection as well. The way to fix this if the functionality was extremely limited such that only a specific file type can be opened or ran in a specific directory. For this the input needs to be normalized. To prevent multiple commands being ran by not system command reads to start with by using file\_get\_contents() , and prevent folder and directory traversing with basename(); Now the user can access any file in the /configs/ folder but nowhere else. This also addresses the next alert, remote OS command injection.

As you can see from Figure 7, All alerts and security problems have been resolved.

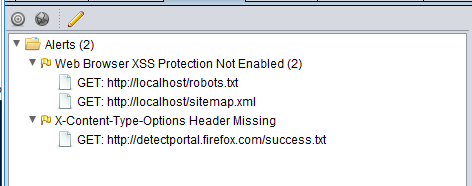


Figure 7- Resolved security alerts