CPSC 42700 Project 5: PHP Web App Exploitation Documentation

**What causes an RFI vulnerability in PHP and what it allows an attacker to do**

RFI can happen when a user of an application can gains control of variables, which then allows the user to execute their own specified code to be executed instead of the code intended by the application. When writing PHP applications there are two main paths that when combined can lead to RFI vulnerabilities inside your applications. The first needed step is a logic error in the application. This is main problem with this type of error is it usually leads to giving the user the ability to be allowed grab variables as part of a request. It’s also important to note the we turned on allow\_url\_include in the php.ini. So, when combined with our previous logic error the problem evolves into an RFI vulnerability.

RFI is a very dangers vulnerability to have on exposed servers. This type of vulnerability gives the attacker the ability to execute their own code on the web server. With this ability they can use it to do tasks such as generating shells, running code, execute a through post-exploitation, and could even elevate their privileges. So, you can see It can be very damaging if a serer has this kind of vulnerability**.** In our mock scenario of the RFI exploit, we were able to not only steal a user’s cookies, which is already a dangerous ability; we were also able to grab the entire password list for theDVWA server. With the extra information you could even conduct a further attack, causing long lasting damages

**A description of the RFI vulnerability that this project is exploiting in the DVWA code.**

The whole point of this project is to exploit anRFI vulnerability that is present inside the Metasploitable’s DVWA server. We are able to use this vulnerability to execute our own php code inside the server, as if the code was there from the very start. With this newly injected code we can then read the stored variables inside the server, including the current users’ cookies and the server’s passwords. Once we have that captured data, we can shunt it over to our attack server. At this point the attack is complete and we have successfully exploited the DVWA code. We were able to steal very valuable data and transmit them back to a safe server for later use.

**The steps of the exploit as they are carried out by your code when the user clicks the bait link, including the order in which scripts are executed and the flow of data between them.**

The first step in our attack is you get the user to go to our mock html page. We would store this code on our own server and run it as a website. The html doesn’t hold much data, just the correct URL with the added pointer for are postinfo.php file. In our example once the user is on our site, they will then click a hyperlink taking the to the main part of the attack. The link will send the user towards the server with the RFI vulnerability (The Metasploitable’s DVWA server), along with a pointer to our attack code stored on our server. Once the DVWA server receives the users request, it while then go to our server using the pointer we added to the hyperlink and will fetch the postinfo.php code (http://10.0.2.15/postinfo). Once the DVWA is done fetching and receives our php code, the vulnerable server will then run it. The postinfo.php in now running on the DVWA this code will pull two pieces of data. The user’s cookies and the servers’ passwords. Once we have “pulled” these files we then will send them back to our attack server via the saveinfo.php. This works because the postinfo.php code uses curl to send the data to another bit of code we have, called saveinfo.php. The saveinfo.php code is stored on our attacker server where it will save the specified data received from postinfo.php. This saved data will then be appended into pre-setup files named “stolencookies” and stolen “stolenpasswds” which are located on the attack server.

**A description of were each piece of code is stored, where it ends up being executed, and where its data comes from and is sent.**

There are three main pieces of code in relation to this attack. The first code is piece is the **baitlink.html**, which is hosted on our server as a website (*it doesn’t have to be the same attacker server, but we still have to host it*.). This code is extremely a simple html page and almost entirely consists of just a hyperlink that leads the user to go to the DVWA vulnerable server fi page, along with the added pointer of our Postinfo.php.

The next bit of code is the postinfo.php, this gets hosted on our attack server and is meant to be injected into the DVWA server using the RFI vulnerability. Even though this code is stored on our attacker server it will get grabbed and executed by the DVWA server. Once executed it will send two data files; a passwd file, and the cookies for the current user. This data will get sent back to our attack server using curl.

The last section of code that’s involved in this attack is the saveinfo.php file. This is code is both hosted and executed on our attack server. This is because it only receives exfiltrated data as POST variables all of which comes from the postinfo.php code. Once the data is received it’s then saved them to files on the attack servers’ local disk.