Yamin Mousselli

Target 1 Epilogue

1) account.php is the vulnerable PHP page. More specifically, the vulnerability exists in lines 19 and 24.

2) I was able to determine the value of response by using an incorrect response value. You're not supposed to tell people the correct password or response code if their credentials are invalid. In account.php, they compute the response from the account number, routing number, and the challenge (this is done on a php compiler). One can easily use a php compiler to compute their response code by taking the output of $teststr on a php compiler.

Additionally, checking the response code alone is not a great way to authenticate a user. The code should check the source of the request.

3) Patch the vulnerabilities mentioned in line 1, use a CSRF framework, and introduce CSRF tokens.

Target 2 Epilogue

1) index.php is the vulnerable PHP page. More specifically, the vulnerability exists in line 32.

2) The vulnerability on the server side exists because the code echoes whatever the user enters in the form without checking or sanitizing the input.

While the vulnerability exists in the server's code, the bug actually originates from the user's browser, not the server. In other words, the HTML and JavaScript code generated by the PHP code for the user's browser is where the issue lies. The PHP code gets executed on the server side while the HTML and JavaScript gets read and executed in the browser (follow the DOM hierarchy). You'll be sending the user's credentials after the user clicks the Login button. The username and password are not sanitized. This all happens at the browser and involves modification of the HTML and JavaScript that defines the web page along with its behavior. However, the PHP vulnerability is more severe than what is going on the client side.

3) Patch the vulnerability mentioned in line 1 and sanitize both username and password on the client side before passing it to the server. Authentication should always occur on the server side.

Target 3 Epilogue

1) auth.php is the vulnerable PHP page. More specifically, the vulnerabilities exist in lines 45, 57, and 58.

2) Line 45 doesn't work because custom filters do not work. Hackers have been creative enough to manually bypass these filters (e.g., stripping). Therefore, the developer failed trying to filter the username. This is what causes the vulnerability and enables attackers to succeed in their SQL Injection attempts. Moreover, the developer needs to change the hashing algorithm used for passwords because md5 hashes can easily be cracked.

3) Patch the vulnerabilities mentioned in line 1 and properly sanitize the username and password on the client side before passing it to the server. Use a strong hash algorithm like SHA512 for password hashing. Authentication should always occur on the server side. Use prepared statements.