**Archivr Solution Steps**

1. Discover that /index.php is vulnerable to Local File Inclusion, allowing reading of webserver source code through PHP filters:

/index.php?page=php://filter/convert.base64-encode/resource=index

2. With the source code obtained through LFI, discover /classes.php.inc which includes a custom PHP class, DirLister, for working with filesystem directories. Note that this class implements the magic PHP \_\_wakeup() function, which will be invoked when the object is constructed through unserialize(…). In the custom class, this function will echo the contents of the directory the object is configured with. Given this, it should be concluded that the challenge might require exploitation of a serialization vulnerability.

3. Given the existence of file upload functionality in the application, it should be thought that the serialization vulnerability might exist there. Recent research (<https://cdn2.hubspot.net/hubfs/3853213/us-18-Thomas-It's-A-PHP-Unserialization-Vulnerability-Jim-But-Not-As-We-....pdf>) shows that a serialization vulnerability exists in file operations on PHAR archives. Conclude that a malicious PHAR archive can be uploaded and exploited through the previously discovered LFI vulnerability.

4. Construct and upload a malicious PHAR archive with metadata containing a DirLister object that lists files of the webserver’s root directory.

5. Using the returned retrieval key and previously obtained source code, determine the path to the uploaded PHAR archive.

6. Use the previously discovered LFI vulnerability to invoke deserialization of the PHAR metadata. Note that a “/” is appended (used to reference specific files in the archive) to circumvent the appending of “.php” found in the source code. The final exploit URL should look like:

/index.php?page=phar://uploads/<dir\_hash>/<file\_hash>.phar/