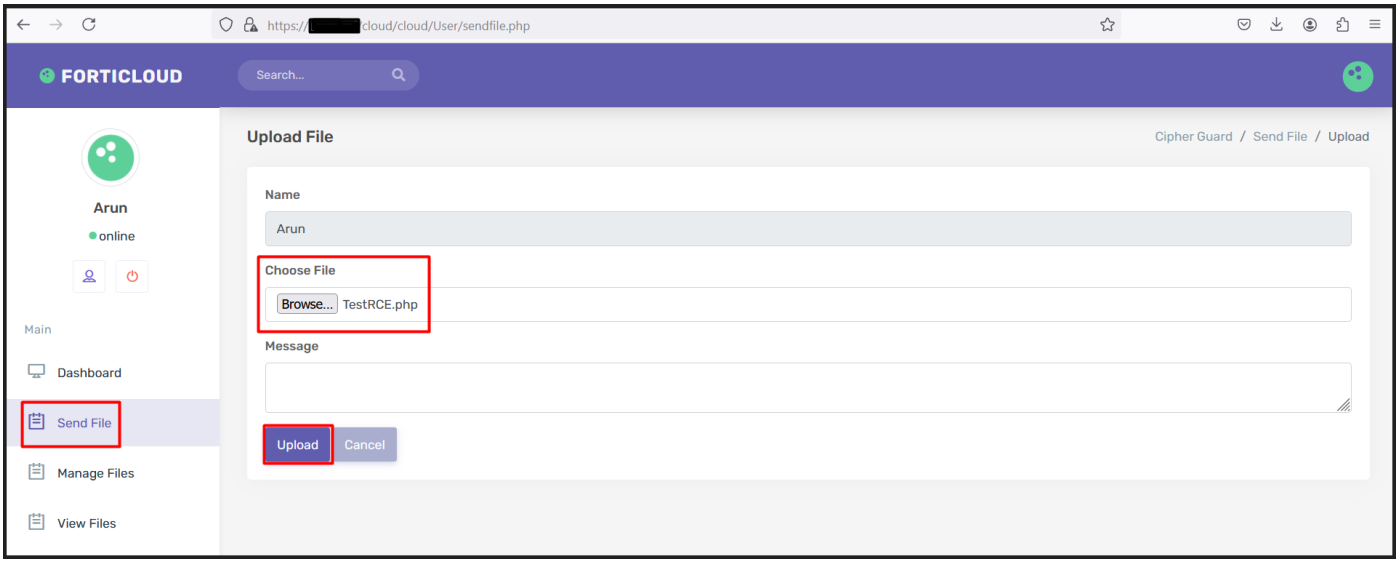
Remote Code Execution through File Upload

Description: Command injection is an attack where an attacker runs arbitrary commands on the host operating system through a vulnerable application. This happens when the application passes unsafe user data (like forms, cookies, or HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. These attacks occur mainly because of poor input validation.

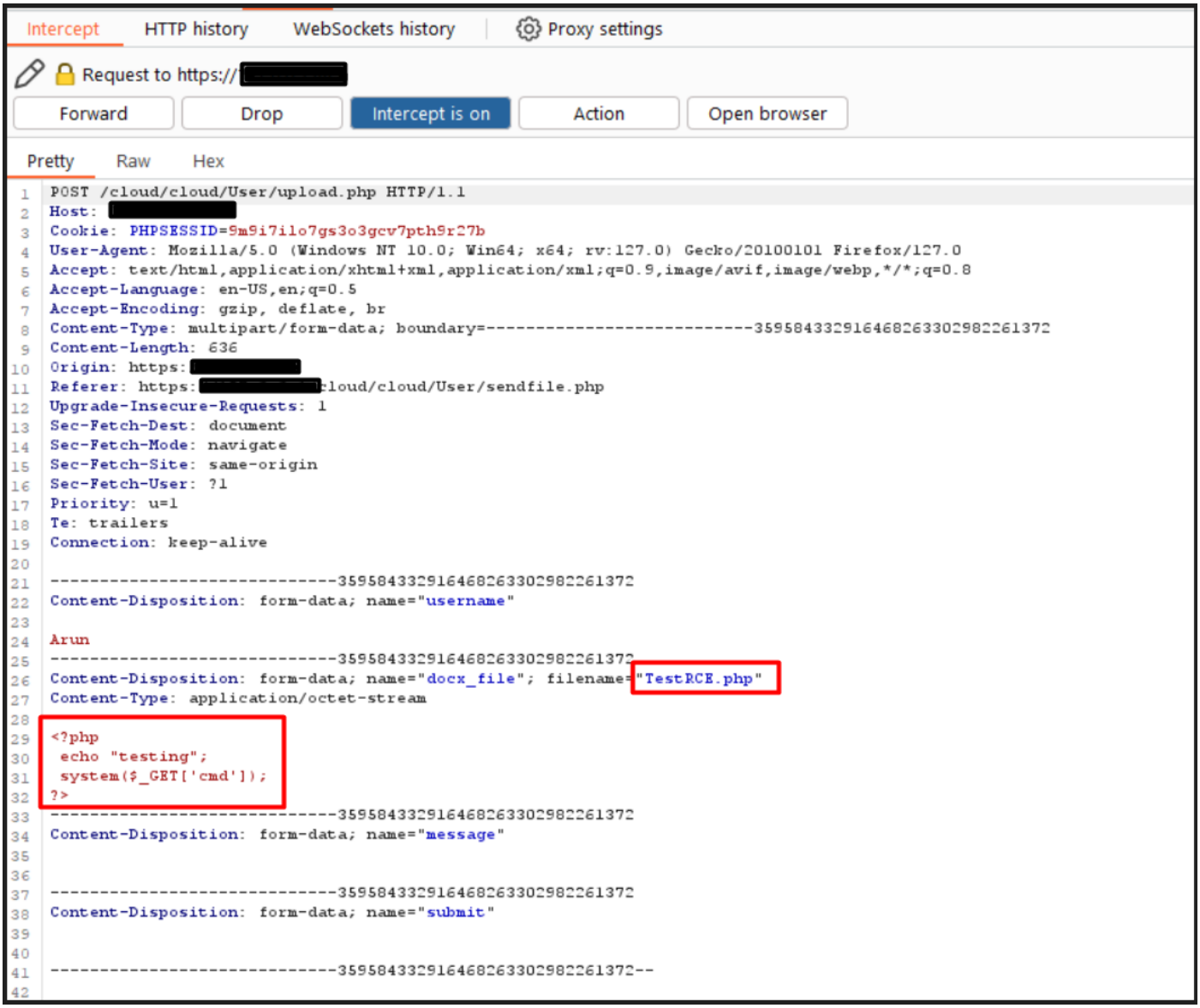
The consequences of unrestricted file upload can vary, including complete, execution of remote commands system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file, especially where it is stored

Testing Process:

Step1: Please navigate to the **Send File** module and upload a .php file, which containing malicious PHP code (refer to Picture 2).

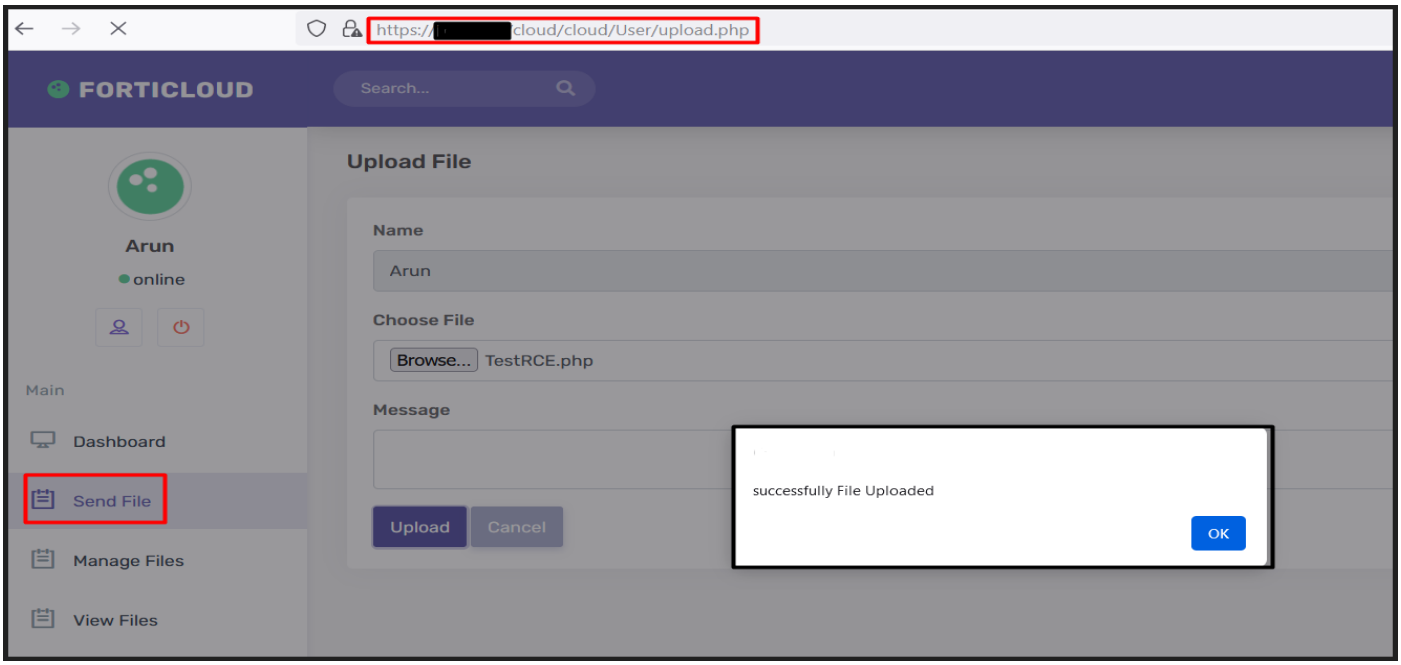


Picture 1

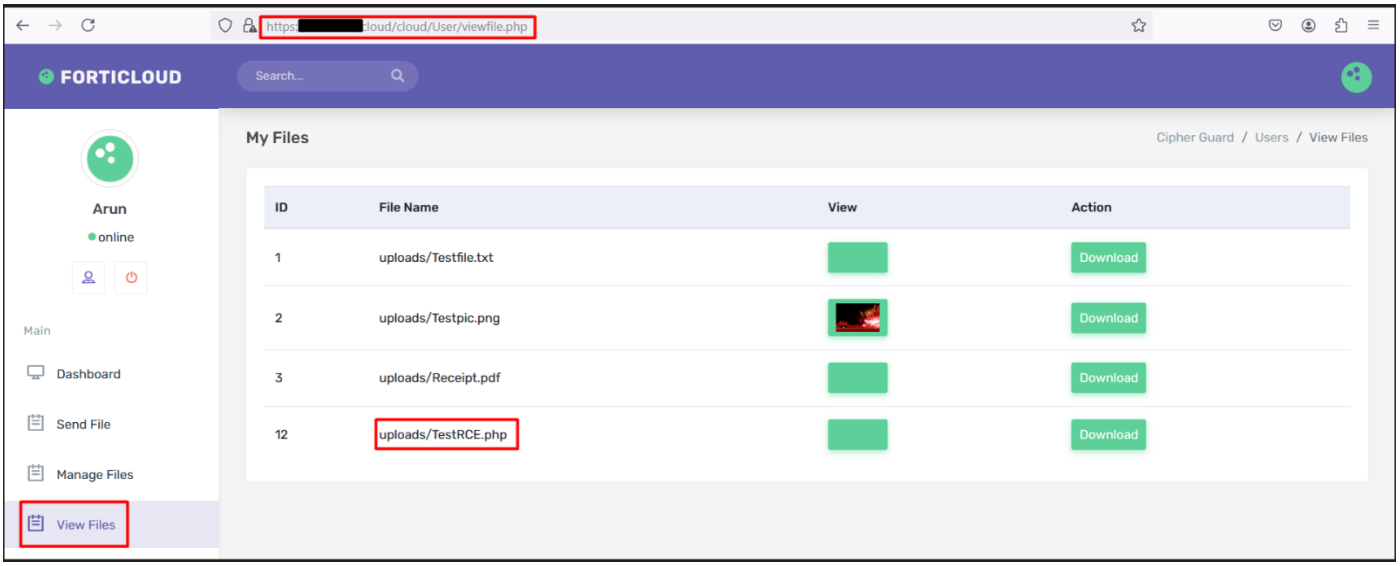


Picture 2

Step2: Now, we can see that the file has been uploaded successfully. To confirm this, go to the **View Files** module, where you can see the uploaded files.

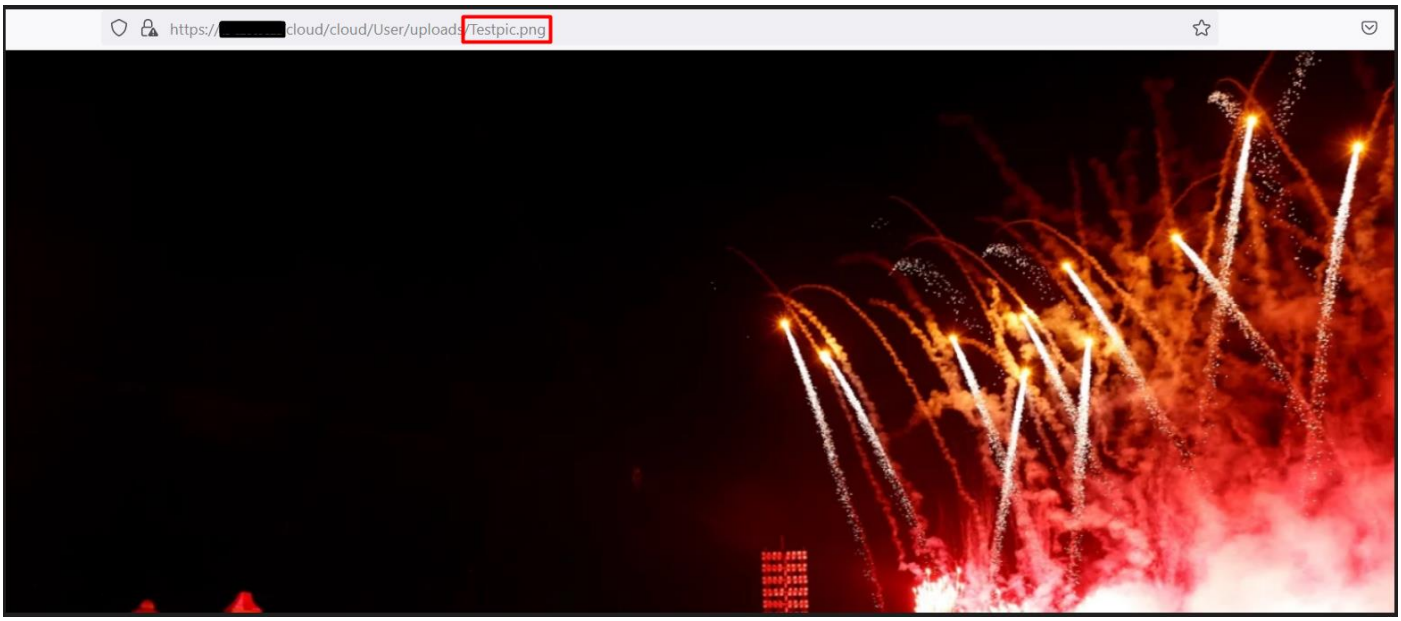


Picture 3

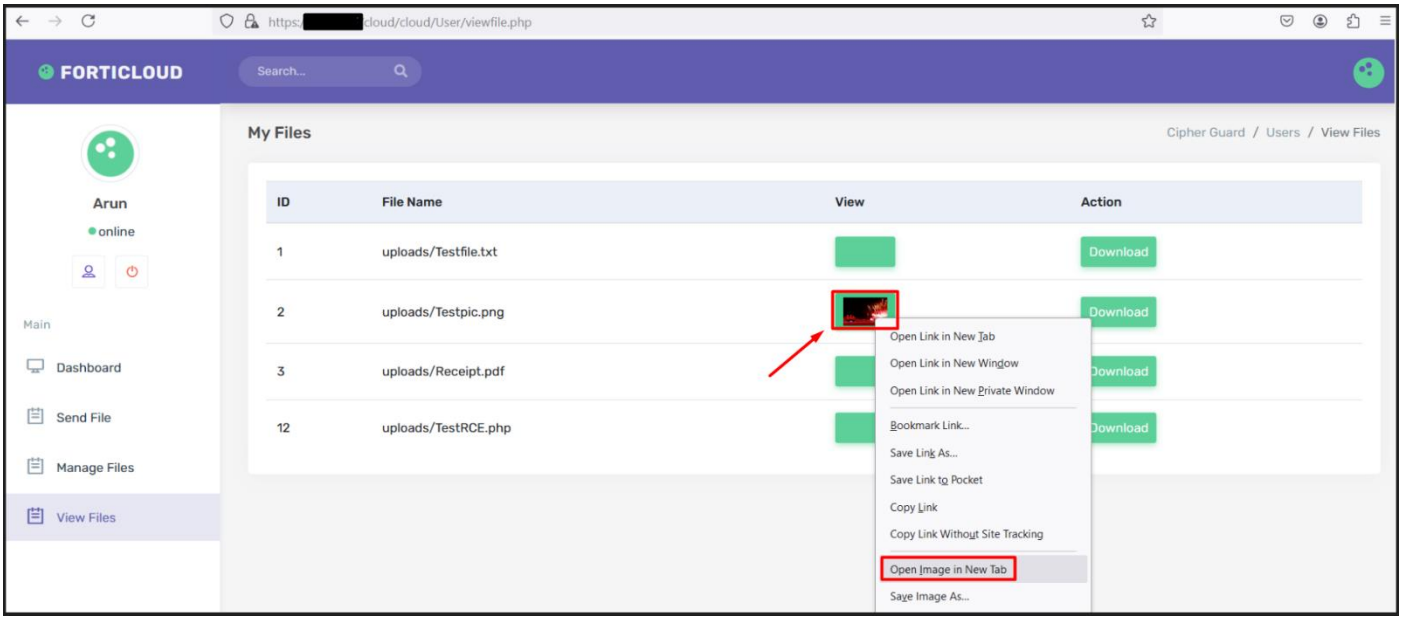


Picture 4

Step3: Now, right-click on the image and select the 'Open image in new tab' option. Here, you can observe that the filename is reflected in the URL.

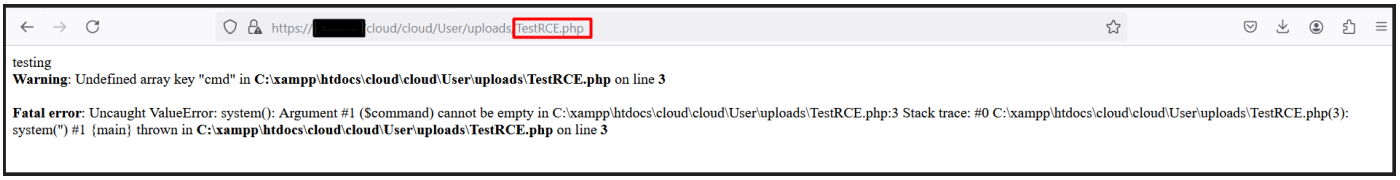


Picture 5



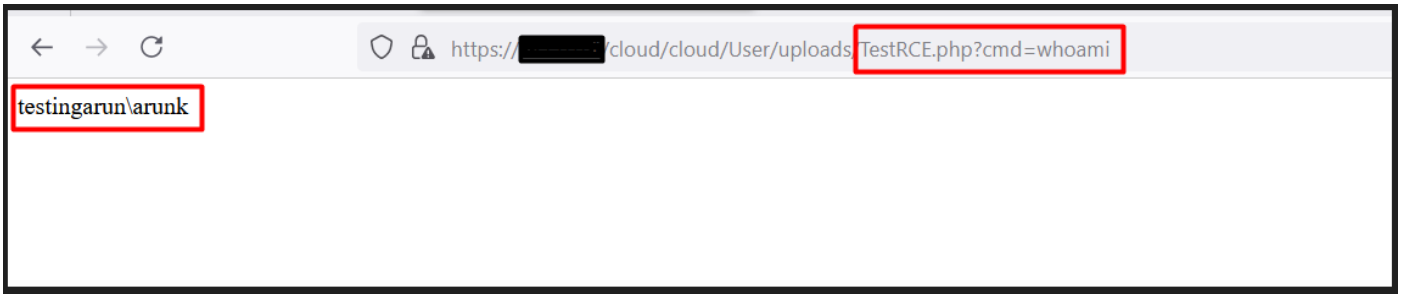
Picture 6

Step4: Now, try to open the malicious PHP file that we uploaded. To do this, change the filename in the URL from **Testpic.png** to **TestRCE.php**



Picture 7

Step5: Now, append **?cmd=whoami** to the end of the URL. We will observe that the remote code execution has been successfully performed.



Picture 8

Remediation: It is recommended to implement the following.

1. Ensure that thorough validation is implemented to verify the file type during the upload process.
2. Deploy a mechanism to detect and reject malicious files during the upload process.
3. Enforce server-side sandboxing measures for all files uploaded to the system.
4. Apply strict restrictions on all file types and known threats like viruses and ransomware, verified through file signature checks.
5. Conduct a file extension check on each endpoint where files are uploaded.
6. Refrain from using shell execution functions unless absolutely necessary, and restrict their use to specific, essential scenarios.
7. Execute comprehensive input validation procedures when integrating user input into shell execution commands.
8. Employ a secure API method for accepting user input within the application.
9. Escape special characters in situations where using a secure API method is not feasible.

Reference:

https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

https://cwe.mitre.org/data/definitions/77.html