

Vulnerability Report

Prepared by

Iulia Mihaiu

iulia.mihaiu@student.unitbv.ro

June, 2021

1. Challenge and scope description

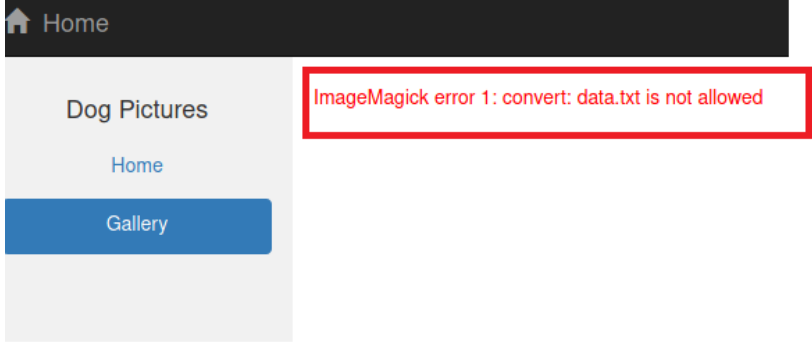
The challenge I had to solve was a capture the flag challenge called "Dog Pictures". From the description of it I gathered the following information: every uploaded file is converted to *.jpg* and the location of the flag is at *"var/flag/flag.txt"*.

Thus, the main goal of the challenge is to upload a file in the web application in order to be able to retrieve the contents of the *flag.txt* file.

2. Identified and exploited vulnerabilities

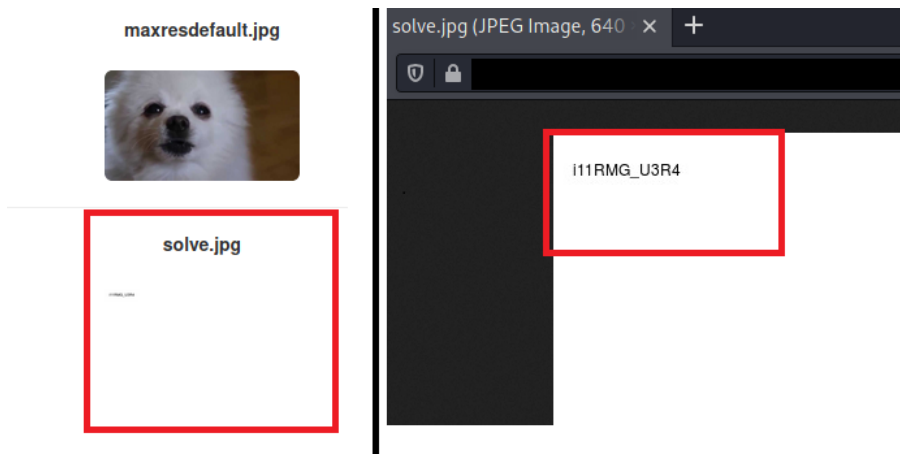
In the following subchapters, the weak points and the findings will be documented in detail.

2.1 Information Disclosure

Severity	Medium
Description	<p>Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users.</p> <p><i>References:</i> https://cwe.mitre.org/data/definitions/200.html</p>
Evidence	<p>When trying to upload a file that doesn't have the <i>.jpg</i> extension, the server returns an error which is disclosing the software doing the conversion of the uploaded file (see the following screenshot).</p>  <p>In this case, the software is Image Magick, which is known for a lot of vulnerabilities.</p> <p>An attacker, knowing this information, can easily look up the possible vulnerabilities of the software and find ways to exploit the web application.</p>

Remediation	In case of an error, the application should return a generic message, informing the user that an error occurred. If it is necessary to record debug information for support or diagnostic purposes, this should be held in a server-side log that is not publicly accessible.
--------------------	---


2.2 File content read in Image Magick

Severity	Medium
CVSS Base Score	7.1
Description	<p>The content of a file can be read using an exploit that contains a specific protocol or command (eg: the "label:@" protocol) which can be executed by ImageMagick.</p> <p><i>Impact:</i></p> <p>Exploiting this vulnerability may allow an attacker to read limited arbitrary files, yet it does not result in an output file that is controlled by the attacker.</p> <p><i>References:</i></p> <p>https://www.cvedetails.com/cve/CVE-2016-3717/</p>
Evidence	<p>In the screenshot below, you can see that using the "label:@" protocol in an exploit uploaded as a .jpg, I managed to obtain an image showing the contents of the "/var/flag/flag.txt" file.</p> 
Remediation	Patches have already been released for remediating this vulnerability, therefore it is recommended to upgrade

	<p>ImageMagick to the latest version.</p> <p>It is also recommended to edit the etc/ImageMagick/policy.xmlfile and disable the processing of MVG, HTTPS, EPHEMERAL and MSL commands within image files. In the section, the following lines should be added:</p> <pre><policymap> ... <policy domain="coder" rights="none" pattern="EPHEMERAL" /> <policy domain="coder" rights="none" pattern="HTTPS" /> <policy domain="coder" rights="none" pattern="URL" /> <policy domain="coder" rights="none" pattern="MVG" /> <policy domain="coder" rights="none" pattern="MSL" /> </policymap></pre>
--	--

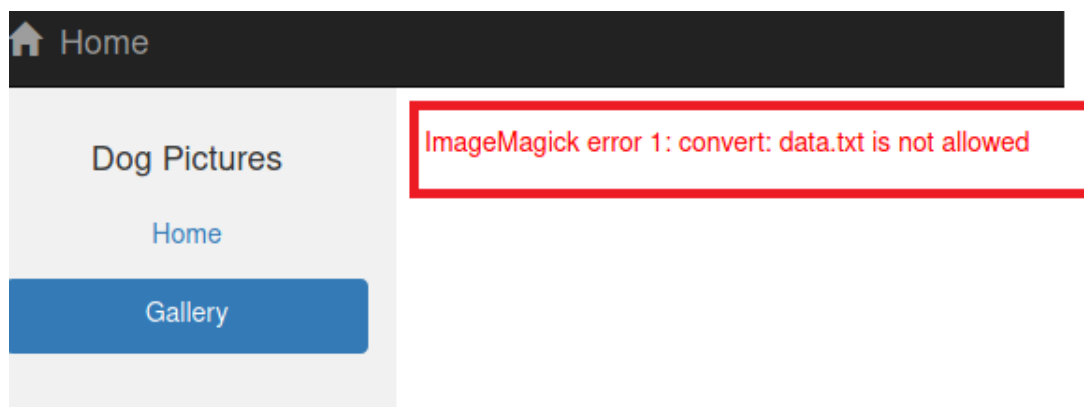
2.3 Unrestricted File Upload

Severity	Medium
Description	<p>File upload is a vulnerability which allows attackers to upload malicious programs or scripts and even their own shells in order to be able to execute them remotely.</p> <p>Attackers can upload code to delete or modify the contents on the server or even create a persistent backdoor and then send the link of the uploaded file to the client so that the file is executed on the browser, by this being able to even create a backdoor on the client side as well.</p> <p><i>Impact:</i></p> <p>Uploaded files might trigger vulnerabilities in broken libraries/applications on the server side (e.g. ImageMagick flaw that is called ImageTragick!).</p> <p><i>References:</i></p> <p>https://owasp.org/www-community/vulnerabilities/Unrestricted File Upload</p> <p>https://cwe.mitre.org/data/definitions/434.html</p>

Evidence	<p>I created an exploit in order to read the content of the "/etc/passwd" file and I saved it with a .jpg extension, which I then successfully uploaded on the web application. Below, you can see that the image in the gallery corresponding to the exploit is a .jpg showing the content of the file we wanted to read.</p> 
Remediation	<p>Verify that uploaded images begin with the expected <i>magic bytes</i> corresponding to image file types before these are processed. This is to ensure that the <i>images</i> being uploaded are actually images, and not exploits.</p>

3. Challenge write-up

The first step that I did was to check if I can upload a different type of file than a .jpg. It turned out that indeed, I can not do that, but the error that I got was disclosing the software which was converting the uploaded files.

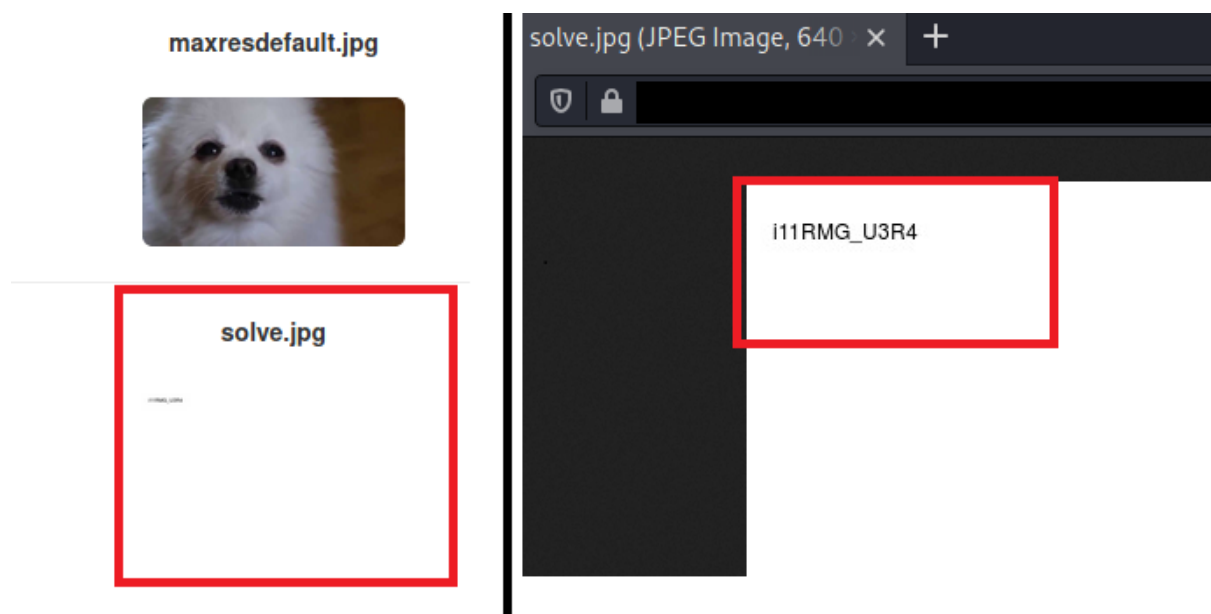


Therefore, I started looking for ImageMagick vulnerabilities and exploits that might work. From the 5 main vulnerabilities that I have found, I chose the one for the file content read, using the "*label:@*" protocol in the following exploit:

```
solve.jpg
1  push graphic-context
2  viewBox 0 0 640 480
3  image over 0,0 0,0 'label:@/var/flag/flag.txt'
4  pop graphic-context
5
```

I saved it as *solve.jpg* and I successfully uploaded it to the web application.

Below, you can see that in the gallery there is now an image that corresponds to the *solve.jpg* exploit. Opening the image we can see that it shows the contents of the file we wanted to read (*flag.txt*).



4. Conclusions

When having an web application you should be careful how much information you are revealing to the users through the errors. The error messages should be as generic as possible.

Another thing that should be frequently done is to check for software updates and patches in order to keep your data away from possible exploits.

Moreover, just checking for the extension and content-type is not enough, you have to filter the actual content and if in doubt, it is better to not accept the file.