Attacking optical character recognition system

Author: Vishwaraj Bhattrai

Email: vishwatek.sb67@gmail.com

Twitter: vishwaraj101

13/08/2017

Summary

Optical Character Recognition(OCR) is the process of electronically extracting text from images, documents and then reusing it in a variety of ways such as full-text searches, invoice processing, Identity or KYC verification process. This one such use case will be harmful when such extracted texts or results from the uploaded document are being reflected within the application or being stored persistently within the database without any input validation.

We will see one such example of an **XSS** attack out of many possible attack scenarios.

Let's explore, as an attacker we need to prepare an image containing our **XSS** vector which if the OCR parser reads and reflects back to the user will allow our malicious content to be injected within the application.

We will take sample jpg as an example

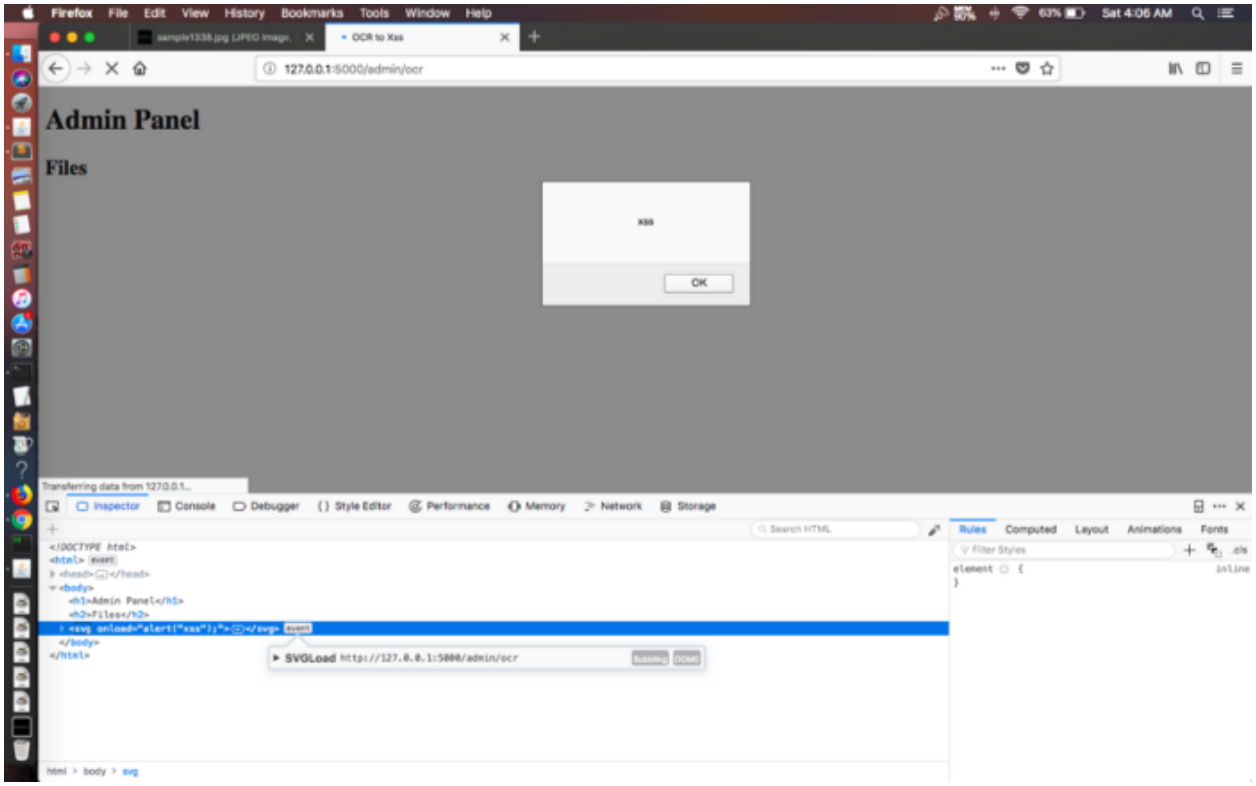
We can create an image like this from here.

Poc

I am using tesseract here for OCR along with a simple flask server which accepts the image as an input and it parses and reflects back the extracted content to the admin or another user. We can find the code here.

Steps to reproduce the scenario

1. First clone the repository from here
2. Start the program by running >>> ***python ocr.py***
3. Now visit the local server 127.0.0.1:5000 from the browser.
4. Upload the above poc jpg file, then visit the directory ***/admin/ocr/files*** and we will see the alert.



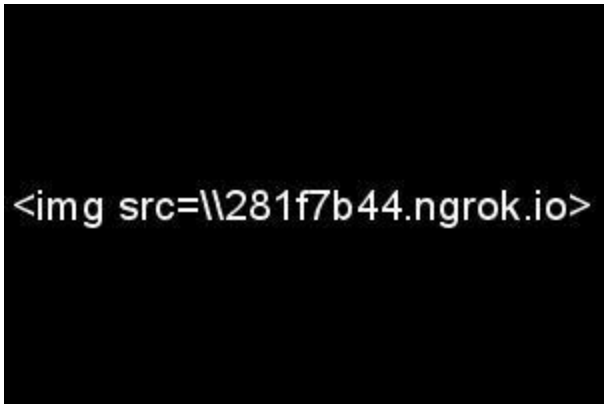
Triggered XSS

Similarly, create an image with the blind XSS payload to confirm a pingback to your local server.

Note

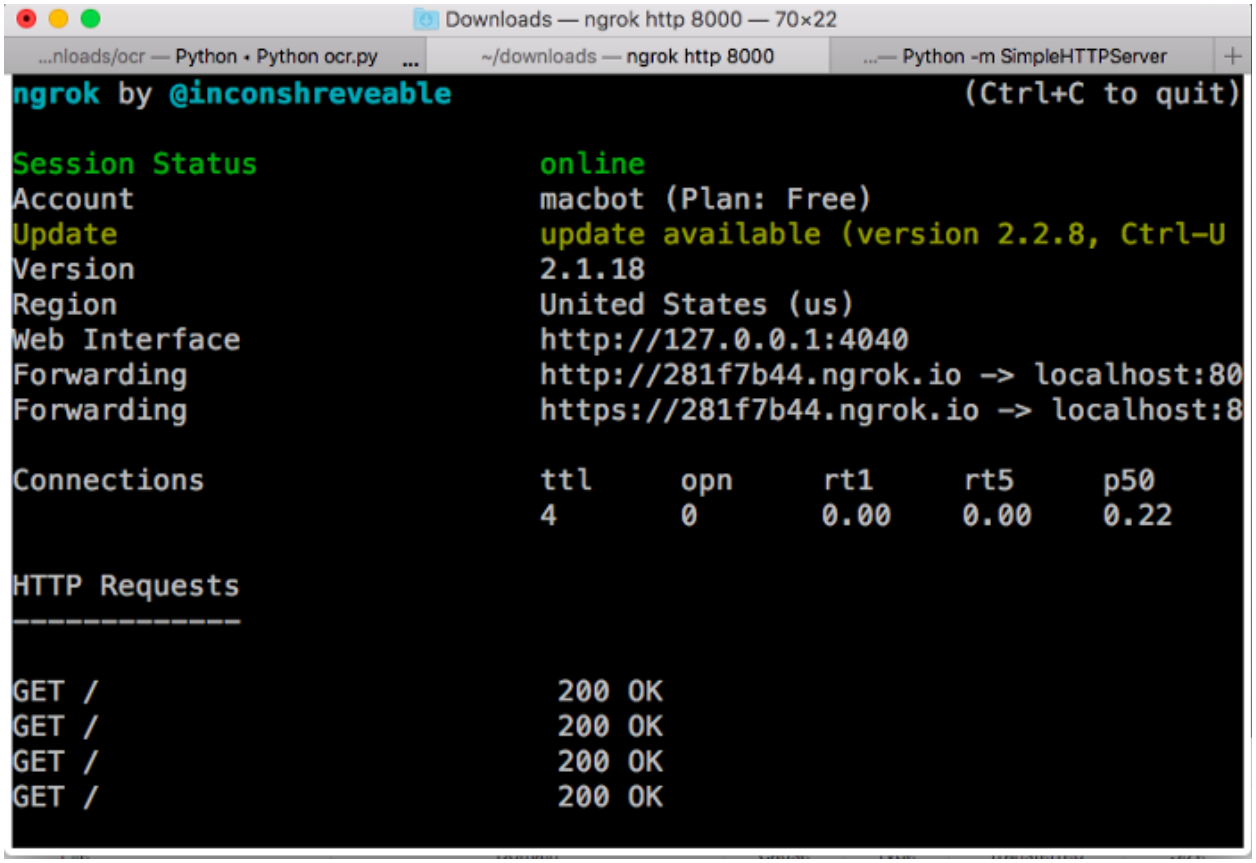
Different parser acts differently to some character, like tesseract will treat forward slash “/” as L so when you will put http:// it will become http:/l because of which it won’t work in the browser so for that I am using backslashes. like that, we have to figure out for other parsers too.

Here I am using ngrok.io just for confirming the ping, we can also use burp collaborator or any other tool. So create your image with the below content then upload and see if you get any hits on your server.



Upload the image with Blind XSS payload

Response



Ping received

Mitigation

If you are using an OCR service then not only filename but also sanitize the extracted text, links from the image or documents before storing them into the DB.

Once you upload an image, check whether the contents or metadata of an image are also reflected within the application. If there is no check on how the output text is being reflected then it's easy to inject the malicious content. Also if the content is stored then this could also allow us to achieve **SQL injection** and other vulnerabilities in **second order context**.

So next time when you see any application asking for KYC or for uploading scanned documents, passport size photo, document verification you can test the system with this approach.
