Lab instructions

Q1:

For this lab, we have opened our own Husky Banking website. The website is available at the domain name bank.com which is mapped (by DNS) to the IP address xx.xx.xx.xx. It uses port xx, which is the default web port; a port is a 16-bit integer identifier of TCP or UDP socket in the computer, allowing the same computer to hold multiple separate TCP/UDP connections (each using a different port). Each student will find a (userid, password) in the file Q1login in the Lab4 folder. Open our bank site and login to your account and note the balance(s).

Q2:

In file Q1 you will find a third username, beginning with the letter V; the letter V stands here for Victim, and in this question, you will expose the password of that poor, careless user. You could do this `manually', by testing common passwords; and, indeed, the user's password is one of the passwords in the file Q2dictionary, containing a `dictionary' of common passwords. However, that would be dull, tedious work. Instead, write a small Python script to try all possible passwords. For that you need to learn a bit about the HTTP protocol, the operation of this website, and about Python client-side web scripting. It would be more interesting than trying all passwords, and surely quicker, and would serve you well in this lab, this course and beyond.

Q3:

So, in this lab, we will build a phishing website which will masquerade as our `Husky Banking' website. For this purpose, we have installed in your VM the Flask web micro-framework, which is a Python module for developing web applications. You are encouraged to also install Flask on your own machine, allowing you to do much of the work of the lab locally, and to develop your own web application / website. In this question, you are only required to do a simple webpage that will display your name. This would be similar to the Flask minimal application; see also in the Flask Mega-Tutorial.

Q4:

In this question, create your first phishing page. This page will look like the real Husky Banking website, with the same background, organization and a form for the user to enter

userid and password. However, your form will not verify these against the legitimate userid-password pairs; instead, you will save this exposed information in a file, and finally, to reduce suspicions, you will automatically use these credentials to login the user into the real `Husky Banking' website, by redirecting the browser. To redirect, you can use the flask.redirect function. Note that Flask may send the HTTP response status code 302, and you may need to override it for correct behavior on all browsers. To show your work, add an `management page' to your phishing server site, where you'll show immediately whenever a new userid-password is collected.

Q5:

We made you a `customized' version of the Husky Banking website. You access this page from a button in the `regular' login page, located right below the `Sign In' button. This `customized' version uses simple JavaScript to `hide' the location of image(s). You should find the location(s) – preferably by reading and understanding the code (there are other ways, we know). Once you have the location(s), modify your phishing website to mimic this `customized Husky Banking' website.

Q6:

Time to do some JavaScript of your own! In this question, you're asked to improve your phishing page, using JavaScript. Specifically, your goal is to record the user's credentials (username and password) immediately as the user types them, without waiting for the user to submit the login form. This is since some users may fill in all or some of these fields, yet decide not to submit the form, e.g., suspect the site just before submitting. As a good phishing attacker, you want these partially filled credentials.