CSE 3140 Lab 4: Web Programming and Phishing Attacks

Created: Fall 2022, Updated: Spring 2024

Monday sections: submit by 03/31

Wednesday sections: submit by 04/02

In this lab, we will learn some basic **Web programming** and **Phishing** attacks. In a phishing attack, users are lured to login at a 'spoofed, imitation' website, which masquerades as a web-site trusted by the users. When the users do not notice that this is a fake website, they may enter credentials (e.g., password) into the spoofed site, which will be captured by the attacker, or download malware since they trust this phishing website. Phishing is a surprisingly effective and apparently the most common way to break into user accounts; it is usually performed indiscriminately against random users, but also launched 'tailored' to specific victim users, in which case, it is referred to as **spear phishing.** Phishing is one of the **social engineering attacks**, which exploit vulnerable human behaviors; other social engineering attacks include shoulder-surfing and other exploits of insecure password practices, SIMjacking (tricking phone company to port phone number to attacker's SIM and phone) and other impersonation attacks, malicious-captcha (tricking user into disclosing private information via fake CAPTCHA), catphishing (luring user by fake/fictional identity in social network) and more. Beware!

Grading: in this lab, the total number of points is over 100. Points over 100 may be ignored or may have some impact toward the final course grade. *Unlike previous labs, please find the rubrics for Lab 4 on HuskyCT, which can be used as the guidelines for your submission and grading.*

Question 1 (5 points)

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 172.16.48.80. A backup server is at 172.16.48.90. Both use port 80, 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).

From the `red network' within the lab, you would be able to access these websites by typing in your browser URL line the address (URL) http://bank.com , http://172.16.48.80 or http://172.16.48.80:80 (as port 80 is the default web port, you don't have to specify it). This was using the 172.16.48.80 server; you can similarly use the backup server, 172.16.48.90.

When connecting using the VPN, the firewall allows only traffic to specific ports, most notably, port 22 which is connected (only) to the SSH server. So, when connecting using the VPN, we cannot communicate directly to port 80 where our web servers are listening. Instead, you should reach the web servers using SSH local forwarding, aka SSH tunneling. See instructions in Lab 0; notice you will need to adjust them to connect to this server.

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).

Submit in HuskyCT and submission server (submit.edu): userid and the balance on that account. In HuskyCT also include screen recording of the process.

Question 2 (20 points)

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.

So… The HTTP protocol is a rather simple, textual (human-readable), stateless protocol, which is described in many places; in particular, we recommend the well-written overview of HTTP in the Mozilla Developer Network (MDN) website. HTTP is mainly used for communication between browsers and webservers; the browser sends *HTTP requests* and the server responds with *HTTP responses.* Whenever you fill the login page and submit a guess for the password, the browser uses the fields you enter to compose an appropriate HTTP request, using a very simple encoding to send what you entered in the userid and password fields to the server. The structure of the request is very simple, and you can easily find that it uses the GET *HTTP method*, and how the two values are encoded into the *path* parameter of this GET request. You can find these details in different simple ways.

A recommended way to find the encoding, is by reading the contents of the login webpage, as received by the browser (which the browser than uses to construct the page as seen by the user). The login page, like any other webpage, is sent to the browser in a simple human-readable *markup language* called HTML (Hypertext Markup Language); the MDN website also offers an excellent introduction to HTML, as well as numerous other sources. The basic HTML defining the webpage is sent by the server in a file we refer to as the *base HTML* of the page; a page may contain other objects, including `frames' which also contain additional HTML – but the login page contains only one frame and few images, and of course the login form. It's a simple HTML page; it may be instructive for you to view and understand it, and this is one good way to find and understand the encoding of the login form userid and password fields in the HTTP request. The browser user interface allows the user to view the base HTML; for example, in the chrome browser, you can right-click on the page and then select *view page source* (or simply hit the key combination CTRL + u). The MDN website also has a good explanation of the

Form element (<form>), which is the central part of this page, as well as of the other elements in the page. Notice that forms can be submitted using either GET or POST request.

Ok, so let's proceed once you found the parameters of the GET request sent by the browser upon submission of the login page – using in the above method or another, e.g., using the browser's developer console/tools. Now you are ready to send the same request from your client-side Python script, using different passwords, and process the responses to detect the correct password. To write this script, we recommend the Python *Requests* module, also described nicely in the W3schools website. It shouldn't be hard.

Hint: the website requires the payload to include the key pair value "submit" : "submit". You may want to append this key, value pair to the end of the payload.

Submit in submit.edu website: the password you found.

Submit in HuskyCT: the password you found, as well as your Python script (as text within your report) and screen shots of the main steps you took.

Question 3 (10 points)

Some users use more secure passwords; and many websites take measures to foil attempts to try an extensive number of passwords, e.g., locking-out an account after some number of attempts. Therefore, attackers use additional methods to expose passwords and to otherwise obtain unauthorized access to the victim's accounts, as we will learn in the reminder of this lab and in the *cross-site attacks* lab. In this lab, we focus on the most common – and, conceptually, the simplest – attack: *phishing using a spoofed website.* A spoofed website is a website designed to mislead users by mimicking a legitimate, trusted website; the trusting users then provide to the spoofed website their credentials, download from it software (which will be malware) or otherwise trust misleading and harmful content (disinformation). A phishing attack involves some way to lure users into the spoofed website, such as using *spam* (in email or messaging systems). Both spam and disinformation are important cybersecurity threats, but unfortunately beyond the scope of this course; discuss them with the lecturer or other cybersecurity faculty.

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 team number and names. This would be similar to the Flask minimal application; see also in the Flask Mega-Tutorial.

Submit in submission site (submit.edu): not required.

Submit in HuskyCT: your code (Python script and HTML) as text within your report, and a screen shot of your webpage. **Please also refer to the rubrics.**

Question 4 (25 points)

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.

We recommend you work separately on the different challenges involved: a webpage with a form to save userid and password; *redirecting* the browser to the real Husky Banking website; and replicating the `look' of the Husky Banking website. But we're sure you'll get over all of these challenges without many difficulties; go Huskies!

Submit in submission site: not required.

Submit in HuskyCT: your code (Python script and HTML) as text within your report, a screen shot of your webpage, and a recording, as separate file, of the entire `spoofed login process', including showing the newly collected userid-password pair. (In Windows, use Windows-Alt-R to record screen.) **Please also refer to the rubrics.**

Question 5 (25 points)

A real website is often significantly more complex than our Husky Banking website and will be harder to mimic well; the attacker may need to analyze and `reverse-engineer' the HTML of the page and often also some of the scripts used in the page. There are many other scenarios in web-security where attacker and defenders need to analyze and `reverse engineer' scripts in webpages. These scripts are mostly written in *JavaScript*; we recommend both the MDN introduction to JS and the W3Schools introduction to JS.

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.

Submit in HuskyCT: the url(s) of the image(s), a screen shot of your webpage, and a screen recording (not video) of the entire `spoofed login process'. **Please also refer to the rubrics.**

Submit in submission site: the url(s) of the image(s).

Question 6 (25 extra points)

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!

You can do this using JavaScript. Your JavaScript will need to handle keyboard events, as explained in the MDN site (or in the JavaScript Tutorial (JST) site and other places), and to use the Fetch API, as explained in MDN (or in JST and elsewhere). You will also need to handle the Fetch API in your Flask server. Keep your Javascript in a separate file from your HTML file.

Submit in HuskyCT: your code (HTML, JavaScript and Python), as text within your report, a screen recording (not video) showing a user filling in the user-id and the password fields but NOT submitting the form, and how the server is still learning the password. **Please also refer to the rubrics.**

Submit in submission site: not required.

Note: Please retain copies of your website for possible reuse in the following lab.