Final DREU Project Overview

Jessica May—Princeton University, June-August 2017

My DREU project was called STORMSHIP(s), which stands for Smart TOol for Revealing Malicious Scripts Hidden In Plain sites. Only about half of websites use HTTPS connection and encrypt their web page's traffic. HTTPS uses SSL or TLS protocol which provides the websites data confidentially and protects the clients from malicious attackers that want to interfere with their traffic. However, HTTP isn't secure like HTTP is and STORMSHIP(s) deals with that. It focuses on creating a system that will check the integrity of an unencrypted web page and notify clients when their HTTP connection has been tampered with by a man in the middle attack. The goal of this project is to help users to know when their HTTP traffic has been interfered with and do this without requiring assistance from the server side of the website.

There has been previous work on the subject (linked below) but that researcher created a way to make sure no changes happened by comparing the server side versus client side. However, this project was heavily focused on the website publishers side and helping them to know when their clients were being affected by in flight changes. In STORMSHIP(s), we're trying to cut out needing the owner of the websites input and instead making it focused only on the client's side of things and notifying the clients when malware is found in the webpage to protect themselves against possible viruses.

The project began with gathering Alexa's top 100 sites and loading the webpages in various countries using Amazon EC2. Various locations are being used because we want to see how the web page response differs when it's loaded in America versus China, Korea, Germany, New Zealand and so on. Once the HTML files were loaded in the different locations, a script was created to decode the files into a readable file and then extract the scripts from the file so that a web page's scripts from one location and its scripts in another could be compared. After the script tags are extracted, they were run through a comparison script that first removes identical script tags from two location's files. After all the identical scripts are removed, the left over scripts were compared and their similarity was calculated. Scripts that were over 90% similar were then removed. When web pages are loaded in different locations, frequently, the country code or language code will be different along with user ID's. By checking the similarity percent, the program accounts for small changes like this.
After almost identical scripts are removed from the diff, the scripts are then split up line by line and similar lines are removed. This removes all singular identical lines and leaves behind the different lines which can then be fed into the classifier to determine whether the change between the two locations is malicious or benign.

Then, malicious sites gathered from an online database were looked at to find similarities between infected websites to know what to look for to determine a website as malicious or benign.

There was an attempt at adding a deobfuscator to simplify and make purposefully messy code readable to feed into the classifier that would be made. Unfortunately, the best deobfuscator found did not change the majority of the malicious websites. Also, when the deobfuscator was used, errors would commonly be produced. One website worked successfully in the obfuscator, but since only one out of nine websites worked correctly, it was not added to the program as it could cause it to crash due to the amount of errors thrown.

Next, a keyword search was created to find web APIs that commonly showed up in malicious scripts and was adapted to include words and functions not previously there. Once a decent list was made, the HTML code from a malicious website was run through the keyword search to see which keywords were found and returned. If the website came back with 0 or little keywords found, the keyword search was adapted to include the keyword of the malicious section of that website. Once the keyword search had more breadth, it was then used on the top 20 of Alexa's sites. Most websites came back negative for any keywords being found, but a few returned positive for some keywords. Since sites that were being considered benign were returning positive, the search needed to be adapted.

During the last week, I created a naïve classifier with 9 malicious webpages and 9 benign (Alexa). It started with testing thresholds of frequencies of the keywords found to see if there was a certain amount that indicated whether a site was malicious or benign. However, the benign sites used in the test return either 0 keywords found or upwards of 20 found. While malicious sites return around or below ten. Due to this type of naïve classifier not working, a n-gram classifier was then created. The malicious websites were looked at to find common patterns between keywords that appeared together. Once the common pairs were found, the new search was run on the 18 HTML code files. This classified 2 benign site and one malicious site as the wrong category and giving an 16% error rate.