# Homework 3 – Requesting Data

## Lab Information

### Due Date:

Homework 3 Dropbox Deadline

### Objectives/Goal:

To evaluate common security concerns using a custom high-speed user-agent.

Hooli Inc. is a reputable web application service provider. This semester you will be serving as Hooli's Web Application Tester.

By the end of this week, we will complete our unit on HTTP and underlying web technologies. At this point, you should be able to understand and automate a large portion of the activities you undertake on a daily basis. Your goal will be to use this knowledge to further your pentesting of Hooli. After you have developed your solutions you will be charged with scanning the Hooli infrastructure.

**You must use your socket based agent from the previous exercise**

### Deliverables:

- Your Docker files
- Your writeup
- You outputted files

### Table of Contents:

# Activity 1: Extracting Data

HTML is a powerful markup language that is used in many different ways throughout the web. Because of this you often need some context as to what you are grabbing. Hooli wants to make sure you are up to the challenge of scanning their network so they want you to demonstrate you understand basic HTML processing

## Step 1: Teaching your agent – HTML Processing

Because of its versatility, processing HTML really requires a parser. Fortunately for you, the BeautifulSoup library, or your language's equivalent, allows just such a capability. BeautifulSoup allows you to use traverse the document object via various methods including XPATH. Demonstrate that you can use it by automatically extracting all the courses that have both numbers and names from the Computing Security program overview page (https://www.rit.edu/study/computing-security-bs). If a course number does not have a name or vice versa, the data should be skipped from the final output.

This exercise is designed to introduce you to the difficulty of retrieving data from sites that may contain data in non-standard formats. The result should be a CSV that lists ONE course number with ONE course name per row. For example:

| | |
|---|---|
| CSCI-141 | Computer Science I |
| CSCI-142 | Computer Science II |

**You should create a new Docker container that can execute your code. The code should generate a CSV list of courses with BOTH course numbers and courses. The data should come from all the tables on the pages and be presented without leading or trailing spaces.**

## Step 2: Teaching your agent - Threading

While manually going over one page is nice, often you'll have to fetch resources external to your site. It will always be much quicker to parallelize this activity. Modify your user-agent to include threading so you can show Hooli that you can quickly save all the pictures of people from https://www.rit.edu/computing/directory?term_node_tid_depth=4919 into a folder.

**Provide a separate Docker container so that it can execute this updated code. This code will be run 100 times (by me) to ensure that it can download all the images on the page. Before submitting ensure that all the images on the page are being correctly downloaded.**

# Activity 2: Crawling for emails

While you were busy demonstrating your capability to quickly download photos, the security team realized that this technique can be used as part of their phishing campaign. They asked you to expand your tool to be able to scan large pages for email addresses.

## Step 1: Teaching your agent - Scope

Alright, alright your agent is getting pretty powerful, but it is still only limited to processing one page, websites consist of hundreds of pages. Modify your user-agent to be able to crawl an entire website. To crawl a site you may wish to familiarize yourself with a few search algorithms such as IDDFS/BS/DFS. Use these search algorithms in conjunction with the links on each page to generate a tree to crawl. Your code should allow you to limit to a domain and a numerical depth of search. You may also wish to add the ability to scope based on URL path.

## Step 2: Teaching your agent – SSL/TLS

You are going to need to interact with Hooli's ultra-secure website. In order to do that you'll need SSL/TLS to be supported. Implement SSL/TLS support within your user-agent.

## Step 3: Crawling for emails

Now that your user-agent can use BeautifulSoup, is threaded, supports TLS, and you can limit by scope – you should be set to crawl for email addresses. Use your user-agent to scan https://www.rit.edu to a depth of 4, for emails.

Depth here should be taken to mean number of links away from the root node (https://www.rit.edu). For example https://www.rit.edu would be depth 0, and any link on that page (for instance https://www.rit.edu/about-rit) would be at depth 1. You can quickly see how MASSIVE this scan is going to become, so allocate time appropriately to run this. You should limit your search to the rit.edu domain.

**Create a new Docker container that it can execute this new code. Generate a file for each depth with the emails in it. Print one email per line in your output. You must match 1000 of the first 5,000 emails that are discovered in order to pass this activity.**

# Activity 3: Harvesting URLs

Hooli has challenged you with doing some recon their site. You know like many sites they often have goodies hidden in places that are not hyperlinked from the main site. You want to create a scanner for these, but before you can you need to create a list of where things might be hiding. Use your new spidering capabilities to generate such a list.

## Step 1: Teaching your agent – Multiple Targets

You've done well, now you need to do more than just one domain. In this case, we're not looking for email addresses but actual paths. While there are existing dictionaries of common directories, for our

next step we're going to need to build our own. Use the same concept as your previous activity to visit at least 25 sites on the companies list provided (companies.csv). Visit them to a depth of at least 4 and record their URLs

### Step 2: Extracting Data

Now that you have thousands of URLs you should extract the data you need (the path without any parameters). In the next task you will use this list to find hidden directories and pages. For this task parse your URLs into an output file such that it can be used as input for the next stage. Output here should only contain unique directories, and should be broken up by level. For instance, the following URL http://www.example.com/javascript/test/potatoe/1.js should generate the following:

```
javascript/
test/
potato/
1.js
```

**Create an output file (named paths.list) with one path per line. Provide the file and your code to complete this activity.**

## Activity 4: More Intelligent Brute Forcing

Hooli understands security, and they want only their engineers with their applications to be able to access the next flag. To that end, they've put a little challenge in place that only their application can solve. Can you get by it and get the secret message?

### Step 1: Scanning for gold

Using your new list of possible paths and pages, scan the new Hooli website http://csec380-core.csec.rit.edu:83 to see if you find any folders or files that were previously not available from the homepage. You may wish to also spider the page such that you can brute force known existing folders. Note down any of these files that might represent security issues in your writeup.

*Hint: Consider how response codes can help you know if paths exist. Using this knowledge consider that you may have to iterate over your list many times to fully spider a site.*

### Step 2: Writeup

- Is there certain information about the webserver that you can discern based on what files you can access?
- Are there any ways to improve the speed of your scanner?
- How can response codes be used in order to more efficiently search your site?
- Are there any common naming patterns that you might expect would yield positive results?

**Provide a file output of the paths that you found on the provided site. Additionally, provide the writeup specified.**

Signoffs

## Activity 1.1 – Provide your Dockerfile that scrapes the CSEC site (15%)

## Activity 1.2 – Provide your Dockerfile that scrapes photos (15%)

## Activity 2 – Provide your Dockerfile that scrapes email and an output email list (25%)

## Activity 3 – Submit file output + code (35%)

## Activity 4 – Submit file output and writeup (10%)