When it comes to browsing the internet, it is clear to see that most, if not all websites are rife with a variety of different links. These links have a very broad range of applications, such as redirecting to a relevant webpage, providing images, as well as advertising products, just to name a few. With anonymity and privacy on the internet becoming further out of reach as time progresses, It spurs a curiosity in what these links are doing exactly. For example, there are 'spy pixels', which is a concept that was talked about in class, which are links that contain essentially invisible images that are used for the sole purpose of tracking user information. (Osborne, n.d.). With the thought in mind that there exist potentially malicious links that remain hidden within the millions of other links across websites, it poses the question; how many links do the websites of organizations actually contain, where are they redirecting to, and are they actually relevant to the webpage that they are on? With answers to these questions, A lot can be learned about the purpose of links across the internet, and how to recognize what they are doing. Additionally, understanding various sources and re-directories embedded in websites gives a better understanding of potential adversaries or other security risks that compromise user anonymity. Starting off with the methodology of our analysis, We needed to create a web crawler that would allow us to see the information that we were looking for. We needed a way to easily view the amount of links on a webpage, as well as other supporting information, like seeing where these links are connected to, and being able to distinguish their relevancy. We also had to make the decision on whether or not to use a proxy for our web crawler. To begin with, we needed to figure out specifications on how to build the web crawler. We used a Firefox Webdriver for our project because there were some websites that would not load unless Firefox was used as an implementation. With the Firefox webdriver we could easily set up an ad blocker as well, which would help in cases of filtering out unnecessary links. In terms of using a proxy, we had to go through a bit of testing to decide whether or not we wanted to use one. After having trouble with

attempting to implement one, and difficulties importing the module, we ultimately decided against using a proxy.

First off, when using the web scraper that we made, it will print out 3 things: the Original Links, which is related in some way to the website link, things such as the subdomain of the website are counted; Filtered Links, where the links are blocked by our ad blocker; the Non-Accessible Links, where the links are not accessible for whatever the reason may be; the Third-Party Links, in which it may or may not be relevant to the website; and lastly, Relevant Third-Party Links, in which the user can input certain keywords to see if the links may be relevant, this can include things like shop, app, store, cloud, etc. We wanted to know if the user experience would be safe when searching on some well-known websites, in which some websites may be using third-party websites as a type of redirection to currency, performance, etc. We imported the library urlparser, which helped check if a URL and the domain of that URL are valid, meaning that the website should have HTTP or HTTPS at the beginning of the URL, that way we know that it's safe since it would be verified. This means that when the user is inputting the website link, it news https:// or http://. We also used selenium and its web browser from Firefox to retrieve the links from the website we were searching and how we could find these links using a tag name. We also used requests and time to make HTTP requests for the adblock rules and time to give it time by pausing the script for 5 seconds, allowing the website to load before we processed the links. Lastly, we used the argparse module for user input in the script so that they can find out what website has third-party websites that are relevant to the original website.

For our approach, we decided that the security of finding these third-party websites would be from the users, in which they would type their keywords in the script, and see if these websites would be relevant to the original website. For example, if we were to put in a website like https://roblox.com, we can see that there are a few websites like itunes.apple.com,

play.google.com, playstation.com, meta.com, etc. These could be relevant to the website, as they might have something that relates to shopping. If we were to re-run the script and put in words that are related to shopping, like store, shop, purchases, etc. It would find those links and see if these third-party websites are checkouts where the user had bought something online with their money. The downside of this is that this only catches the surface-level links that are 'assumed' to be safe by the user's keywords. This means that a link that could be relevant could definitely not be relevant at all, and could probably be malicious. Our code only checks from the user's input and sees if what they want to know is relevant to the website and assumes that it's safe based on the user's assumptions.

Each function in the code is used to verify the website and check the various links that the website has. First, when we were implementing our code, we wanted to make sure that the URL was valid, and was using urlparser. Urlparser helps check if the URL is valid by checking the scheme of the URL and the network location (netloc) of the URL. We also wanted to check if the link domain was valid to its original domain, so if the base domain of a website is still in the same domain as the website, it would be valid and also count towards being part of the original website. Essentially, it would check if the subdomains of the website would still be connected, like facebook.com/home/ is connected to facebook.com. One of the first implementations of trying to find third-parties was where it first needed the user to input both the domain and the website link to check if it was part of the domain. This was later updated so that only the website link was needed instead of having to put the main domain name and the url link into the input. Next, we needed to use an adblocker such as EasyList to catch some malicious links, so if a website were to have a malicious link that EasyList had, it would count it as a Filtered Link, meaning that it would be malicious. This could also be implemented by the user as well if they had a filter list for blocking malicious links. Unfortunately, from many of the websites that we

have gone through, there were absolutely no malicious links that could be found by EasyList. Afterwards, we would process the input of the URL, which basically tells if the user has used a .txt file or a single URL link to look through. If it's false, it goes to the next function, which processes the websites from the file. Note that when using a .txt file, it will make the links related to the website Non-Accessible, since the condition is_allowed is returning false. We are uncertain at this time of why it returns false when running through a text file, but it does still give out the links that are relevant and what might be 'third-party.'

Now, after either processing the website from a file or just processing a singular website, it would then go through our main function, website_crawler_with_adblocker which first initializes a headless Firefox browser, which then navigates to the specified URL, and processes all the links from the webpage. This is where the different types of links are categorized, as mentioned earlier, it checks and prints if the link is original, filtered, is a third party, is relevant (based on the user's input) or is not accessible. We would then use the total number of links to see what could be potentially malicious, or safe depending on the user's context. For example, the user could be using keywords like 'virus' or misspelled words to find websites that might be harmful; or perhaps it could try searching for relevant keywords that might be related to the original domain, such as 'data, store, shop, auction, bids, security, defend, etc.' to see if they would have relevant results depending on the user's needs. That way they could see what third-party websites may be relevant to the original domain. After getting the link(s) from the user's input, it then uses the argparse module—where we allow users that would use the script to execute their command arguments. For example, if a user wants to just look at all the links of the website, it could just use a URL link; if a user wanted to look at relevant URL links, it would type in the link and type in the following keywords that the script would look at to see if it's relevant or not.

Running our code takes in a website URL input and outputs links associated with said URL. Testing with the URL https://www.ebay.com (given our parameter of a five-second timeout), we accessed around 334 links with a total of 91 third-party links. Examining the third-party links for eBay, we found them to be highly relevant to the website and expected given eBay's position as an e-commerce business. For example, third-party links were found related to eBay's social media pages (i.e., Facebook), websites of distributors selling products on eBay (i.e., TCGPlayer), or alternative sites associated with eBay's business or business model (i.e., eBay's career website or eBay Ads). Using a filter, we examined some typical vague keyword configurations associated with misleading or harming users (such as 'here', 'more', 'readmore', 'learnmore', 'link', 'linkto', 'info', as well as '.exe') (Caron, 2017). Based on our test, we found almost no instances of these keywords used in any hyperlink associated with eBay.com. In comparison, an obvious scam site such as https://www.thisisnotasketchywebsite.com/ had instances of .exe links embedded which may be a sign of malicious activity targeting the user. In the case of eBay, it's expected that a publicly traded company would not have any malicious links in comparison to a gag website that is built with the sole purpose of 'pranking' the user. However, it was unexpected that we did not get any filtered links whatsoever, but it shows that eBay uses naming conventions that seek to inform the user rather than deceive. The majority of the links observed were easily identifiable and we could easily predict the outcome of selecting said link.

Through our research, we have been able to get a more broad view on the types of links that can appear on websites. Although our original intention was to be able to easily identify links that are unrelated to the websites they are on, we found it more difficult than expected to find any malicious links hidden within websites for large organizations. We may have been able to find more info on irrelevant links or hidden trackers if perhaps using a larger variety of keywords beyond those we had tried. Despite this we were able to learn some fascinating information about

how some organizations run their websites based on the links that we had found. For example, how the gaming website Roblox contained links to many other organizations, which may mean that a majority of online tracking is done by large organizations in plain sight, rather than something that might seem malicious. Through our research we are able to answer our initial questions. When it comes to how many links are across large webpages, there are frequently hundreds within all of them, many of them redirect to other large organizations, and in terms of if they are relevant to the website, it is entirely based upon the context of said website. For example, something like eBay could contain many links to other companies that may not seem relevant to ebay itself, but is considered relevant for advertising purposes. Overall, we completed our research with a much better idea of the purpose of the ubiquity of links across the internet.