

|  |
| --- |
| **Location-Based Web Scraping with Physical Security Automation**  *Web scraping and area scan with Deep learning classifier for social engineering.*  **Moustapha Isaac Diaby**  CMP320: Scripting Project  2022/23 |

*Note that Information contained in this document is for educational purposes.*

Abstract (not more than around 400 words)

There is a surprising amount of information that one can find through open-source intelligence. OSINT is the practice of gathering this information from the perspective of a pen tester or social engineer that has been appointed to do an analysis for a corporate body.

We will take a look at the making of the “Open-Source Location Fingerprinting” (**OSLFP** for short) program and go over the decisions that went into it such as performance, scraping, APIs used and the process of creating a classification model to detect gates/fences and cameras in the area of interest.

Contents

[1 Introduction 1](#_Toc136359622)

[1.1 Background 1](#_Toc136359623)

[1.1.1 Understanding Open-Source Intelligence (OSINT) 1](#_Toc136359624)

[1.1.2 Understanding Object Detection AI with the You Only Look Once (YOLOv5) Model 1](#_Toc136359625)

[1.1.3 Project Overview 2](#_Toc136359626)

[1.2 Aim 3](#_Toc136359627)

[2 Program and Development 4](#_Toc136359628)

[2.1 Program Agreement Configurations 4](#_Toc136359629)

[2.2 Scrapping websites 5](#_Toc136359630)

[2.3 Scrapping websites for address information 6](#_Toc136359631)

[2.4 google maps API set up 7](#_Toc136359632)

[2.5 Getting mapping Information 8](#_Toc136359633)

[2.5.2 Satellite Imagery 9](#_Toc136359634)

[2.5.3 Interactive Map 9](#_Toc136359635)

[2.6 Identifying Nearby Public Buildings for WiFi-Based Attacks with google maps 10](#_Toc136359636)

[2.7 AI Classifier Model for Identifying Walls/Fences and Cameras: Yolov5 model 10](#_Toc136359637)

[2.7.1 Testing data and labelling 11](#_Toc136359638)

[2.7.2 Classification and training of the model 13](#_Toc136359639)

[2.7.3 Implementing the model in the OSLFP script 14](#_Toc136359640)

[2.8 Reporting the results 14](#_Toc136359641)

[3 Discussion 15](#_Toc136359642)

[3.1 General Discussion 15](#_Toc136359643)

[3.2 Countermeasures 15](#_Toc136359644)

[3.3 Future Work 16](#_Toc136359645)

[References 17](#_Toc136359646)

[Appendices 18](#_Toc136359647)

[Appendix A 18](#_Toc136359648)

# Introduction

## Background

It is surprising how much valuable intelligence can be derived from open sources such as employees’ emails, addresses and even confidential documents. This skill known as Open-Source Intelligence (OSINT) is the practice of gathering this information from the internet. Within the realm of cybersecurity, OSINT serves as a valuable tool for penetration testers and social engineers who are entrusted with analysing the vulnerabilities of corporate bodies. Additionally, with the recent advancements in Artificial Intelligence (AI) there are now greater possibilities to automate I detect threats faster and more efficiently.

### Understanding Open-Source Intelligence (OSINT)

OSINT involves the systematic collection and analysis of publicly available information to gain insights and assess the security posture of an organization. It encompasses various sources, including online platforms, social media, websites, public records, and more. OSINT practitioners adopt the perspective of a pen tester or social engineer, leveraging the available data to uncover potential weaknesses in the target's digital infrastructure and human factors.

It is important to recognise that OSINT plays a pivotal role not only within the cybersecurity community but also within the governmental and intelligence sectors. Esteemed institutions such as MI6, MI5, and various military organizations rely on OSINT as a critical component of their intelligence-gathering and analysis efforts.

Governmental intelligence agencies recognise the value of OSINT in gathering information from publicly available sources to enhance their understanding of potential threats, geopolitical situations, and emerging trends. OSINT serves as a foundational pillar in the comprehensive intelligence framework, complementing other intelligence disciplines such as signals intelligence (SIGINT), human intelligence (HUMINT), and geospatial intelligence (GEOINT).

### Understanding Object Detection AI with the You Only Look Once (YOLOv5) Model

Since This paper is not focusing on AI and machine learning we will briefly go over what Object detection AI, powered by advanced machine learning algorithms, has revolutionized the field of computer vision by enabling automated identification and localization of objects within images or videos. One popularly used object detection framework is the You Only Look Once (YOLO) model, specifically the YOLOv5 variant. This section aims to provide an overview of object detection AI and the key principles underlying the YOLOv5 model.

Object detection AI involves training machine learning models to recognize and locate specific objects within an image or video frame. It enables automated identification of objects, their spatial boundaries, and sometimes their semantic meaning. This technology finds applications across diverse fields, including autonomous vehicles, surveillance systems, robotics, and for this paper, physical security.

The YOLOv5 model is an evolution of the YOLO family of object detection models, known for their real-time inference capabilities and high accuracy. YOLOv5 is based on a deep convolutional neural network (DCNN) architecture and is designed to efficiently detect and classify objects with impressive speed and precision. YOLOv5's architecture is implemented in Python and allows for easy adaptation and local training on specific datasets, making it flexible for various object detection tasks.

**Key Principles of YOLOv5:**

1. **Single-Shot Detection:** Unlike traditional object detection methods that require multiple passes over an image, YOLOv5 adopts a single-shot detection approach. This means that it directly predicts object bounding boxes and class probabilities in a single forward pass through the neural network, resulting in faster inference times.
2. **Anchor Boxes:** YOLOv5 employs the concept of anchor boxes to handle objects of different scales and aspect ratios. Anchor boxes serve as prior knowledge for the network, allowing it to predict accurate bounding boxes regardless of an object's size or shape.
3. **Feature Pyramid Network:** YOLOv5 incorporates a feature pyramid network that enables multi-scale feature extraction. This pyramid structure helps the model capture objects at different scales and spatial resolutions, leading to improved detection accuracy.
4. **Training and Optimization:** The YOLOv5 model is trained on large-scale datasets annotated with object labels and bounding boxes. During training, the model learns to optimize its parameters to accurately detect and classify objects in diverse scenarios.

### Project Overview

In the following sections, we will explore the key aspects of the OSFP program and its application in analysing the physical security of a target location. The report will delve into the decision-making process behind the program's design and implementation, discussing the considerations made to ensure optimal performance and accurate data retrieval. Additionally, the report will outline the methodology employed to create an AI classification model capable of detecting gates/fences and cameras in interest. Through this comprehensive examination, we aim to shed light on the development of an effective OSINT-driven tool for enhancing cybersecurity tools.

## Aim

The overall aim of this script is to be able to create a Python script that will be provided a website which will be scraped for any address relating to the business. The script will then get Open-source information about the location which will then be used to analyse the physical security of the building/offices and points of interest for a pen tester / social engineer that is appointed to perform a blind investigation (where the pen tester is given no information about the company’s infatuation and no initial access to their network).

The required functionalities are broken down:

* Scrap a website for address information.
  + This step will be done by scraping the website and identifying key tags that contain address information such as a postcode or country code such as “UK” for the United Kingdom or “GB” for the United Kingdom of Great Britain and Northern Ireland.
  + The UK will be the scope of this project.
  + This would also include nested links of the initial site and any sub-domains.
* Create and make use of an AI classifier model that will Identify Walls/fences and cameras around the building.
  + The method for this will be to create a model that identified the formatted objects in a series of images in the surrounding location of the target building and generate a report on how many were found.
  + Due to time constraints, this will be a limited model which will have limited performance as we will not be able to detect all the surveillance cameras and barriers in the area. However, should be able to demonstrate the concept.
  + This will be used to indicate of secure the physical poster of the location is.
* Identify and locate nearby public buildings that are within 100m (WIFI connection)
  + This will indicate points of vulnerability as the hacker will be within range to attempt an attack on the company’s network via the WIFI.
  + Maximum of 5 places
* Identify and locate congestion in the road/pathing.
  + These are points that most employees are likely to walk/drive through which may be useful for a sniffing point for access cards and drive-by attacks.
  + This will be done by locating the nearest main road to the target location.

An important mention of the aims of the project is that it will be part of OSINT fingerprinting as the program will only be able to work with information that's already in the public domain. if the area is the off limited such as a government base – there will be no data outputted or information will be scrapped from the internet.

# Program and Development

## Program Agreement Configurations

For the Python script to function as a script with dynamic functionality, we made use of the "argparse" library, which is included in the standard Python library collection. This library enables us to accept arguments passed in from the user when running the Python script, allowing for easy modification of the program's functionality. The following arguments are configurable in the program:

A screen shot of a computer code

Description automatically generated with low confidence

Figure 1 Image of main.py arguments

* “-a”, “—address”: This argument allows the user to specify an address or location to analyse. By providing an address, such as "Bell St, Dundee DD11HG", the script will focus its analysis on that location.
* “-u”, “--url”: With this argument, the user can specify a URL to analyse. Including the protocol (http or https) is required, for example, "--url http://www.abertay.com". The script will then perform its web scrapping on the provided URL(s).
* “-d”, “--depth”: This argument determines the depth of the search. By specifying a numeric value, such as "--depth 3", the script will analyse the specified location or URL along with nested links up to the specified depth.
* “-sl”, “--scanlimit”: The scanlimit argument sets the number of images to perform AI detection on. By default, it is set to 10, but the user can modify this value as needed.
* “-pl”, “--placeslimit”: This argument allows the user to specify the number of places to find within 100 meters of the location. For example, by providing "--placeslimit 5", the script will identify up to five nearby public buildings or points of interest.
* “-c”, “--confidence”: The confidence argument is used to specify the AI detection confidence percentage. By default, it is set to 0.6 (60%), but the user can adjust this value to fine-tune the AI detection process.
* “--no-relm”: By including this argument, the script will disallow scanning nested URLs that are outbound of the root domain. This option can be useful when focusing solely on the target website and its immediate subdomains.
* “--no-vuln”: With this argument, the script can be configured to skip the physical security scan. By default, the script performs the physical security analysis, but including this argument will exclude that step.
* “-gak”, “--gauth-api-key”: This argument allows the user to provide their Google Maps authentication key. It can be added directly as an argument, or an environment variable named "GOOGLE\_MAP\_API\_KEY" can be used to store the key.
* “-v”, “--verbose”: By including this argument, the script will provide verbose output, offering additional details and information during the execution of the program.

## Scrapping Websites

In this section we will go over how the web scrapper was built and the logic behind it. We first make the operation multithreaded as it will speed up the fetching this can be seen on line 156-158 in figure 2. We make use of a thread pool of 10 and run the “processURL” (line 120) while passing it the URL of the entry point along with some meta data. Here we check that the endpoint has not already been scanned by another thread (on line 122). If the endpoint has not be scanned the thread will continue by fetching the site’s html (line 134) and parse it using BeautifulSoup (on line 140).

A screen shot of a computer program

Description automatically generated with low confidence

Figure 2 image of main.py class “OpenSourceLocationFingerPrint” - “findLocationFromURLs” function

Skipping over line 141, the next line we then scan the DOM for any anchor (link) tags to add to our scan which is handled by the “addNewEntryPoint” function. In this code, the “addNewEntryPoint” method checks if the URL belongs to the same domain (realm) based on the sameDomain flag. If sameDomain is True and a relm is specified, it verifies that the URL contains the relm. If the relm is not specified, a warning message is printed, and the URL is skipped. Additionally, the method checks if the URL has already been added to the searchUrlSet to avoid duplicates which is a set data struct for performance. If the URL passes these checks, a new entry point dictionary is created and added to the entryUrl list. Finally, a log message is printed to indicate the addition of the nested URL.

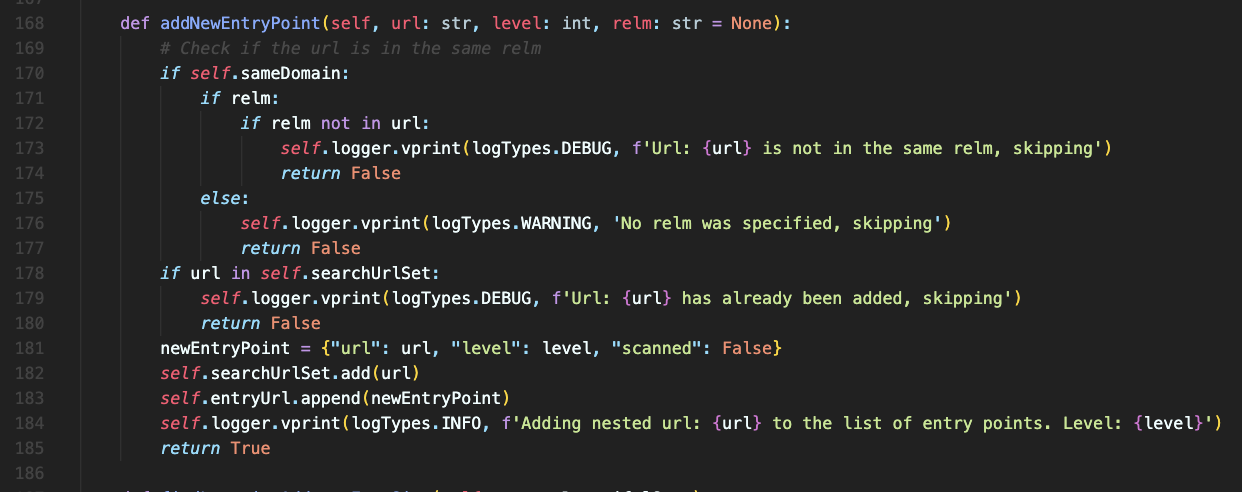


Figure 3 image of main.py class “OpenSourceLocationFingerPrint” – “addNewEntryPoint” function

When the scan depth is met the loop will end as all the possible links have already been checked.

## Scrapping websites for address information

There were many solutions to extract address information relating to the site such as looking up, DNS Recon SOA records, Whois lookup, and integrating Maltego and the combination of them all would be ideal however An easier solution for the proof of concept was to scan the HTML for address tags as they are standard HTML tags for displaying address information.

In the las section, we skipped over line 141 in the main.py file, here we will now go over it.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Figure 4 image of main.py class “OpenSourceLocationFingerPrint” – “findLocationAddressFromSite” function

In this code, the method finds all address tags in the HTML using soup.find\_all('address') (line191 in figure 4 ). If no address tags are found, a warning message is printed. Otherwise, the text within each address tag is cleaned and appended to the addressFound list for scanning and setting it as the target location.

Finally, the code iterates over the addresses found and checks if they are already present in the addressSet again making use of the set data structure for performance and advising duplications. If an address is new, a success message is printed, and the address is added to the set. If the address is already in the set, a debug message is printed to indicate that it has been skipped.

## google maps API set up

The next step was to get the mapping information of the target location and Google Maps API had the capability to satisfy our aims. I've implemented 2 different ways for the user to provide their google maps API credentials (as part of the assessment I will include my own and set it to expire 30 days after my submission). In figure 5 I’ve linked a brief tutorial on how to get your own.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5 https://arc.net/e/2C6BA2AB-400E-4544-9C56-0653DABC446E GCC - Google Maps Set up.

When the user has their own API credentials they will be able to either in input it via the command line with the --gauth-api-key flag or in a .env file with the key called “GOOGLE\_MAP\_API\_KEY”.

We first assign the args.gauth\_api\_key value to self.googleMapsAPIKey. Then, it checks if the key is None. If so, we attempt to retrieve the key from the environment variables using config('GOOGLE\_MAP\_API\_KEY'). On line 248 of Figure 6, we create the googlemaps.Client which is a Python library that wraps the google maps API and allows us to easily interface with it.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 6 image of main.js class “OpenSourceLocationFingerPrint” – “setUpGoogleMapsAPI” function

## Getting Mapping Information

#### Geocodes

Here we will look at how the mapping information is gathered. By using the “googlemaps” Client called “gmaps”. we fetch the geocode for the addresses we found in section 2.3. with this, we can get an accurate coordinate location of the target. The relevant information is then extracted from the geocodeInfoFromGoogleMaps dictionary and assigned to the corresponding fields in self.address, including the formatted address, types, place ID, and geometry (geographic location information). As shown in figure 7.

A screen shot of a computer

Description automatically generated with medium confidence

Figure 7 image of addressHandler.py class “AddressHandler” – “findLocationGeoCodeFromAddress” function

### Satellite Imagery

Next, we can get a satellite image of the area with the googlemaps client. This was done by calling the static\_maps function (as shown in figure 8 line 102) and providing it with the size of the image, centre point, zoom, and lastly specifying that we want wanted the map type to be a satellite image. We receive the image in checks so we must iterate over them so that we can save the full image as original.png

A screen shot of a computer program

Description automatically generated with low confidence

Figure image of addressHandler.py class “AddressHandler” – “getTopDownImageOfLocation” function

### Interactive Map

Lastly, we generate an interactive map that starts at the geocode that we retrieve in section 2.5.1 and use a Python library called folium to generate the HTML code for the interactive map and add markers such as nearby public buildings (more on this in section 2.6).

A screen shot of a computer code

Description automatically generated with low confidence

Figure image of addressHandler.py class “AddressHandler” – “getTopDownImageOfLocationForProcessing” function

In Figure 9 we can see that we retrieve the geocode (latitude and longitude) of the target location on line 112 and generate the map with folium on line 114. We then populate the map with the nearby places that were discovered in section 2.6 and save the map as process.html.

## Identifying Nearby Public Buildings for WiFi-Based Attacks with google maps

Like section 2.5 we can make use of google maps Library and its client wrapper to gather local businesses within a 100-meter radius of the target location with the geocode.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 10 image of addressHandler.py class “AddressHandler” – “findNearbyBuildings” function

In Figure 10 line 127, you can see that we call the place nearby function on “gmaps” and provide it with the geo-location and the radius that we want to search in meters. We then iterate over the discovered places and extract the name, place\_id, types (the type of building it is), geometry (geographical information) and vicinity (the address). This process is continued until the maximum number of places is reached by default its 5 but can be adjusted by the user with the –placeslimit flag.

## AI Classifier Model for Identifying Walls/Fences and Cameras: Yolov5 model

We will now look at how the AI model for the detection of physical security features in the area. The features that we will focus on are gates and security cameras. We will be using the “You Only Look Once” (YOLO) classifier model to achieve our aim.

### Testing Data and Labelling

The first step was to collect the reference images from Google – I collected a series of images (44 for training and 7 for validating) containing fences/gates and cameras to help train the model to identify and classify the features. The names of the filenames were not correlated in a clear format, so I used the following command to rename all of the files in numerical format.

**Command**: ls -v | cat -n | while read n f; do mv -n "$f" "$n.jpeg"; done

A screenshot of a computer

Description automatically generated with medium confidence

Figure 11 Unformatted testing images

A screenshot of a computer

Description automatically generated with low confidence

Figure 12 Formatted Testing images.

The next step was to create and generate an XML file for each image that contains the area where the feature is located this would include the starting x1, y1 coordinates and ending x2, y2 coordinates – to make up an area. To help generate the XML file I used an online tool called “makesense.ai” which can be found here <https://www.makesense.ai/> (Make sense, n.d.)

A screen shot of a computer

Description automatically generated with medium confidence

Figure 13 Image of the MakSens.Ai object detection tool for labelling images

From here we upload our images and select object Detection mode as that will produce the output that is needed for the labelling.

A close-up of a camera

Description automatically generated with medium confidence

Figure 14 Screenshot of the manual object detector

It's important to note when exporting you want to select “export in YOLO format” as shown below.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 15 export format options

When exporting the labels, we get a zip filing containing all the areas we outlined with all the coordinates and classifiers id (0 for gates or 1 for cameras).

A screenshot of a computer

Description automatically generated with low confidence

Figure 16 the output label file for each image file

### Classification and Training of the Model

We used the “You Only Look Once” (YOLO) classifier model to train our AI model. Which we can obtain by from GitHub with the following command: “git clone <https://github.com/ultralytics/yolov5>” This library is built on top of PyTorch (Github, n.d.). We can then use this project as a starting point and configure it to do what we want it to classify. The focus will be on “yolov5/tutorial.ipynb” which is a Jupiter notebook with code to get started. The first change was to point the project to our training data – this can be done by creating a copy of the “yolov5/data/coco128.yaml” file calling it “yolov5/data/oslfp.yaml”then modifying it to reflect the training data and validated data, additionally the names of all our classifiers in the same order that we input in the previous step.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 17 oslfp.yaml file containing information about the testing data

We then ran the “yolov5/train.py” script to train the model with our data with epochs of 200 (Think of this as iterations) as we don’t want to overfit the model with our training data and still let it have time to train.

A screenshot of a computer

Description automatically generated

Figure 18 python train.py --img 400 --batch 2 --epochs 200 --data oslfp.yaml --weights yolov5s.pt --cache

### Implementing the model in the OSLFP script

we will go over the implementation of the model in the script as we have a working model for our use case

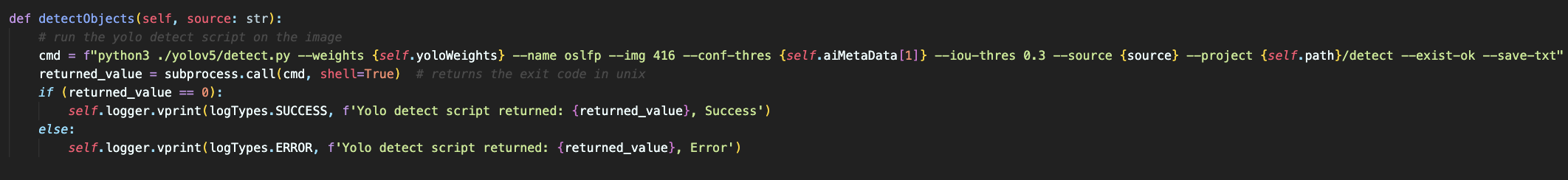


Figure image of imageProcessor.py class “ImageVulnProcessor” – “detectObjects” function

In the above figure we make use of

. detections with above 60% confidence from the model.

## Reporting the results

The script provides the user with a few ways to view and interact with the results it generated. The first is the output of the script. Which goes over the overview of all the findings such as known locations, the path to output files, names and locations of nearby buildings, the output path of the interactive map and the findings of the AI scan for cameras and gates.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure Output of the Result from Script

The code behind this output can be found in main.py line 257 in the generateReport function. It pulls together all of the findings in the previous stages and uses prints and tabs (“/t”) to format the layout and display the results to the user.

# Discussion

## General Discussion

The script is useful….

Usability of the program

In section 2.1 we looked at how the arg parser was implemented - These configurable arguments provide flexibility to the users, allowing them to customize the behaviour of the program based on their specific needs.

In section 2.4 we saw how the Google Maps API credentials are provided into the script - The command line input takes priority as this is the best practice to do so and makes the script easy to use and portable.

Optimisation of code

In section 2.2 we looked at how the web scrapping was implemented – the use of concurrent programming and multi-threading the operation hugely improved performance as multiple entry points could be scanned at the same time.

Ai performance

By using an YOLO model It allows for rapid development with promising results. The model was able to achieve a camera match of 79% and 42% gate match in the training and valuation stage. Note that the model is limited as it has been only provided with only 44 testing images and I was running the model on my own intel-based Mac device, we will go more in-depth in the future work section on how we can improve the accuracy rate of the model. Although the model works to some degree there are a lot of false positive when it comes to detecting gate.

Here, you want to discuss your results/outcomes.

* What does it all mean? Discuss anything of interest. How does this relate to other work in this area (if relevant)?
* Relate the findings back to your aims - how well have you met your aim?

## Countermeasures

Due the project being Open-Source Intelligence (OSINT) based the best countermeasure would be for the target not no have a lot of their physical security online. This may prove to be challenging for companies such as brick and mortar, schools and other buildings that are easily accessible to the public as individuals may post the area on social media not knowing. On the other hand, office builds already have less of a risk for this script to expose all of the physical security measures that are in place around the building, This is because they are not accessible to the public. However, it would be beneficial to train employees not to take images of sensitive areas and share them on social media. An organisation that is remote with minimal cyber presence would be a strong countermeasure against this strip as there will be no information about their HQ, offices, or public buildings within 100m from the target. With that said it is impractical especially for smaller to medium size businesses as they need some level of presence to attract customers.

There are a few ways that hackers could still identify buildings associated with a company and obtain information about the address of the target, one solution (not a feature of this script) is that hackers have other means of identifying such as the company house (an executive agency of the British Government that maintains the register of companies) (Wikipedia, 2023) which is in the public domain that every company in the UK must register and include a registered office address. The countermeasure to this would be to have a mailing proxy that redirects letter mails from one PO box to another – this allows the target not to have their actual address on company house registers. another would be to identify domain ownership records, which can be used to find the locations of servers by using sites like who.is (domain lookup tool/site). The solution to this would be to register with a third party and hide the domain ownership through them so that information about the target company is not directly exposed.

## Future Work

In section 2.3 we looked at the method implemented to scrap for address information from the html, although it fufills the minimum aims for the project it is limited as it doesn’t fully cover gathering the address locations relating to the website. other methods of finding addresses on the site, such as DNS Recon, Whois, or Maltego. You can implement these additional methods to enhance the address discovery process.

The first action would be to improve the Yolov5 model as …

Build upon Maltergo – an OSINT tool that uses a variety of media such as social media, opensource intelligence and private data to generate OSINT reports.

* What would you do if given more time and resources?

# References

Make sense, n.d. *makesense.ai.* [Online]   
Available at: https://www.makesense.ai/  
[Accessed 25 May 2023].

Wikipedia, 2023. *Companies House.* [Online]   
Available at: https://en.wikipedia.org/wiki/Companies\_House  
[Accessed 25 May 2023].

yolov5 , n.d. *yolov5.* [Online]   
Available at: https://github.com/ultralytics/yolov5  
[Accessed 25 May 2023].

# Appendices

## Appendix A

The script is in the LocationFingerPrint folder which will be attached upon submission of this project. Additionally, every file directly referenced has been screen shot and included in the context.