

OSINT Dashboard Project Summary

Andrew Flora
Beacom Institute of Technology
Dakota State University
Madison, SD
andrew.flora@trojans.dsu.edu

William Campbell
Beacom Institute of Technology
Dakota State University
Madison, SD
will.campbell@trojans.dsu.edu

Santiago Colla
Beacom Institute of Technology
Dakota State University
Madison, SD
santiago.colla@outlook.com

Abstract—OSINT Dashboard is a web-based Open-Source Intelligence dashboard, which features tools to analyze a wide variety of publicly available data sources. The target audience of this tool are individuals collecting information about a particular webpage, media, or personal identifiable information, especially those conducting OSINT investigations for potentially fraudulent websites, that may or may not contain media or personal identifiable information.

Keywords—OSINT (Open-Source Intelligence), PID (Personal Identifiable Data), Exif (Exchangeable image file format)

I. INTRODUCTION

OSINT Dashboard is a web-based Open-Source Intelligence tool designed to consolidate and streamline the analysis of various data sources. This tool provides users with the ability to gather and examine information from URLs, image files, and phone numbers.

The OSINT Dashboard will not only collect and present this information but will also provide users with insight on the information that it finds. It aims to simplify the process of gathering data from diverse sources, making it a valuable resource for OSINT investigators and researchers.

II. FEATURES & USER INTERFACE

A. Main Page

The user is greeted with the application's homepage describing each of the tools and their capabilities, along with input boxes for URLs, file upload, or phone numbers.

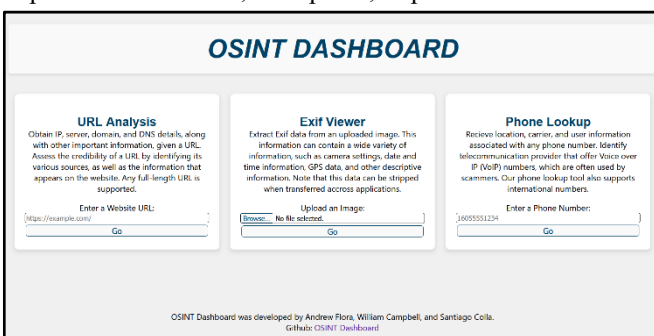


Figure 1: Main Page of OSINT Dashboard

B. Web Analyzer

Within the Main Page, the user is prompted with an input box requesting the URL leading to the target to be analyzed. If the URL is valid and publicly accessible, OSINT Dashboard will extract relevant domain data, such as IP addresses, registrar identifiers, DNS records, etc.

It should be noted that certain websites may block automated requests through their network configuration,

limiting the accuracy of the analysis or even causing HTTP 403 “Forbidden” errors.

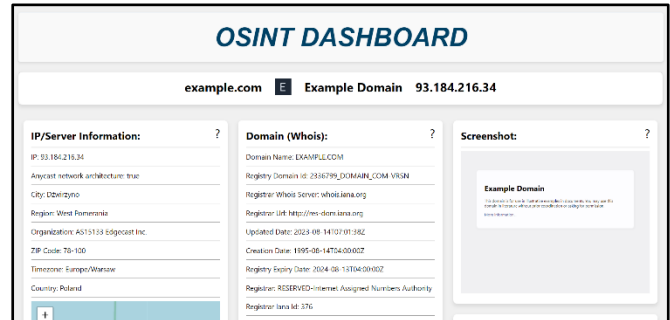


Figure 2: Example Usage of Web Analyzer

C. Exif Viewer

The user is prompted with a file upload input box which accepts .jpg, .jpeg, .png, .gif, and .tiff files as long as they are under 20 MB in size. If the file is not corrupted, it will display all the metadata available. In the case of pictures, location information will also be available as long as the original metadata has not been removed.

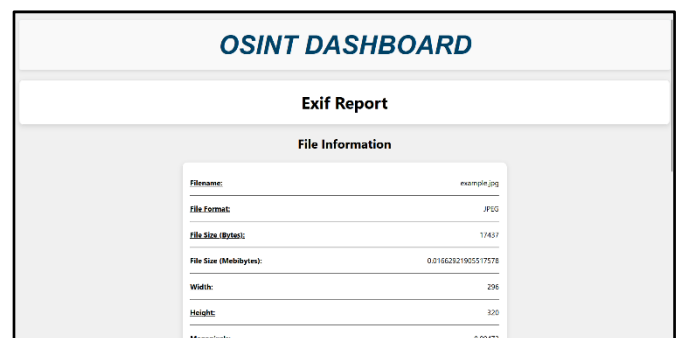


Figure 3: Example Usage of Exif Viewer

D. Phone Lookup

The user is prompted with a text input box within the Main Page that only accepts domestic and international phone numbers; for this the country code is required at the start of every number. The tool will respond by displaying the country, region, validation, and carrier information.

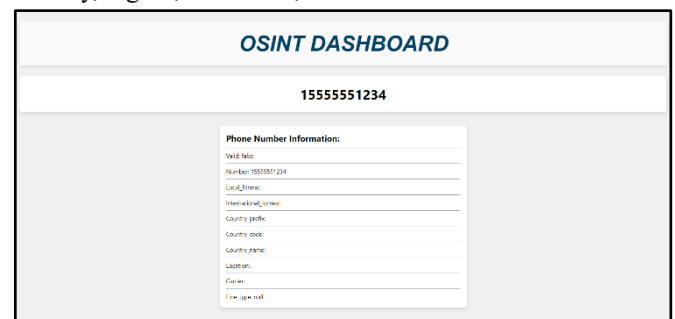


Figure 4: Example Usage of Phone Lookup

III. TECHNICAL IMPLEMENTATIONS

A. App.py

- Our web application comprises three main components: URL analysis, an Exif viewer, and a phone number lookup. We utilize Flask for our backend, with each of these elements having its specific route in which they function.
- The "/web_tools" route is dedicated to URL analysis. Upon receiving a user-provided URL, this route executes asynchronous tasks using concurrent futures to gather information about the input website. The collected data encompasses IP information, cookies, headers, DNS records, SSL information, redirects, sitemap, port information, domain(whois) information, screenshot, link information, email information, and phone number information.
- The "/pid_tool" route manages our phone number lookup functionality. It accepts a phone number as input and obtains information through the 'get_phone_info' function.
- Finally, the "/upload" route oversees our Exif viewer. It accepts a file, extracts Exif data from the image, and displays relevant information.
- All three of these routes packages all relevant information into a JSON file to be sent off to the frontend to be viewed by the user. Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.

B. *Web_tools.py* • The 'web_tools.py' file handles all the information gathering for the URL analysis element of our project. This file contains various function that obtain numerous key details about a user-provided URL. These details range from basic web analysis to information for security assessments.

- The basic functions for our web application deal with the extraction of key information about the given URL. This file includes functions like findTitle to extract webpage titles and is_valid_url to validate URLs using the urlparse method. The website_information function a few basic details about a website, such as its domain, IP address, title, and favicon link. Redirects are handled by the get_redirects function, fetching a series of redirects for a given URL. The file also contains functions that deal with the retrieval of cookies and headers through get_cookies and get_headers functions, incorporating mechanisms to manage timeouts and potential request exceptions.
- This file also includes functions such as get_ip_info utilizing the ipinfo library to fetch information about an IP address. DNS records are obtained using the get_records function, covering various record types. SSL certificate details, crucial for security assessment, are retrieved with the get_ssl function. Sitemaps analysis is covered by functions like get_sitemaps and sitemap_parser, enabling the extraction and parsing of sitemaps from a given URL.

- Additionally, the URL analysis portion of our code checks for open ports using find_open_port and check_ports functions. WHOIS information for a given domain can be acquired using the whois_info function, handling valid top-level domains and providing details about domain registration. To capture a visual snapshot of a webpage, the get_screenshot function employs Selenium in headless mode, converting the screenshot to a base64encoded format.

```
load_dotenv()
ipinfo_api_key = os.getenv('IPINFO_API_KEY')

def get_ip_info(ip_address):
    try:
        handler = ipinfo.getHandler(ipinfo_api_key)
        details = handler.getDetails(ip_address)
        return (details.all)
    except ValueError as e:
        return {'Error': f'{e}'}
```

Figure 5: get_ip_info function with example of environment variable

- Other functionalities include the extraction of internal and external links from a specified URL through the get_internal_external_links function. The script also facilitates the extraction of emails and phone numbers from the text content of a webpage using get_emails and get_phone_numbers functions, respectively.
- Throughout the file, environment variables are used to securely manage sensitive information like API keys. The various functions also utilize robust error handling to ensure even if one function fails it doesn't break the whole system.

C. File_tools.py

- The 'file_tools.py' file takes in a user-provided image and processes the exif data from it using the exifread and Pillow library. It obtains important information such as the filename, file format, file size, dimensions, and mode of the image. Additionally, it extracts Exif tags from the image file, handling GPS information separately, converting degrees, minutes, and seconds to decimal latitude and longitude if available. It loads a pre-existing JSON file (exif_dict.json) containing Exif information, enhancing the collected data. Overall, this script serves to generate a comprehensive report on uploaded image files, encompassing both basic image details and detailed Exif metadata.

D. pid_tools.py

- The 'pid_tools.py' file takes a user-provided phone number and uses the request library to make an API call to apilayer. It makes a get request and receives various information about the provided phone number.

E. Script.js

- The 'script.js' file is used to enhance the overall understanding of the URL analysis by dynamically creating and appending elements based on the received JSON data.. It includes a function called toProperCase for transforming strings to proper case. Another function, createAndAppendElements, iterates over the

provided JSON data, generating HTML elements for the various data tools.

- For each tool, specific configurations are provided, including the container selector, data key, dictionary for mapping keys to user-friendly names, and ignored items.

```
var toolsConfig = [
  {
    containerSelector: ".cookies-info",
    dataKey: "cookies",
    dictionary: {},
    ignoredItems: []
  },
  {
    containerSelector: ".ip-information",
    dataKey: "ip_info",
    dictionary: {
      "ip": "IP",
      "anycast": "Anycast network architecture",
      "org": "Organization",
      "postal": "ZIP Code",
      "country_name": "Country"
    },
    ignoredItems: ["isEU", "country_flag_url", "country", "country_flag", "country_currency", "continent", "loc"]
  }
]
```

Figure 6: Example tool configuration

- The final part of the script, triggered on the DOMContentLoaded event, dynamically populates HTML containers based on the received JSON data, adhering to the predefined configurations for each tool category.

F. Unit Testing

- As each tool within the URL analysis, EXIF viewer, and phone number lookup functionalities is modular and independent of others, we have implemented unit testing to ensure the reliability of each tool individually. For instance, when conducting unit tests for the URL analysis component of our application, we employed the Python library 'unittest.' This involved defining the class 'TestWebsiteTools,' dedicated to testing each function related to URL analysis. The class incorporates a list of 80 randomly selected URLs to thoroughly test each method. These tests are executed using the 'run_tests_for_website' method, which includes assertions to validate the types of the returned results, thereby ensuring the reliability and correctness of the functions.

```
1 import unittest
2 from web_tools import *
3
4
5 class TestWebsiteTools(unittest.TestCase):
6 >     websites_to_test = [...
10
11     def run_tests_for_website(self, website):
12         with self.subTest(website=website):
13             try:
14                 result_info = website_information(website)
15                 self.assertIsInstance(result_info, tuple)
16
17                 result_redirects = get_redirects(website)
18                 self.assertIsInstance(result_redirects, dict)
19
20                 result_records = get_records(website)
21                 self.assertIsInstance(result_records, dict)
22
```

Figure 7: Unit testing URL Analyzer functions

IV. DEPLOYMENT

A. Microsoft Azure Web App

- The flask application was deployed using the free, F1, tier of Microsoft Azure's App Services. The

GitHub repository for this project was connected and authenticated to the Azure App Service resource for a seamless deployment.

- The API keys that were stored were added to the Web App's Configuration Application Settings.
- All features run similarly to the local host version; however, the screenshot portion of the Web Analysis page does not function correctly due to the lack of an operating system container.
- There are timeout errors that may occur due to the lack of resources in the deployed version.
- The web application is hosted at <https://osintdashboard.azurewebsites.net/>.

B. Running Flask Application Locally

- In order to run the flask application locally, all files should be installed within a single directory, along with Python version 3.11 or higher. Install all required python modules using "pip install -r requirements".
- Create a file named .env and move it to the same directory as all the application files. The .env file holds all the API keys and looks similar to text featured in Figure 8.
- Once all preparations are met, you can run the application by running app.py. This will start the development server, which you will be able to access at <http://127.0.0.1:5001/> in your web browser.

```
IPINFO_API_KEY=12345678987654321
SECRET_KEY=12345678987654321
NUM_API_KEY=12345678987654321
```

Figure 8: .env file layout