InfoHound tool

Improving OSINT open source CyberArsenal for good

Xavier Marrugat Plaza

Master's Degree in Cybersecurity and Privacy

Supervisor
Jordi Guijarro Olivares
PRA
Víctor García Font
13/06/23

Universitat Oberta de Catalunya

Universitat Oberta de Catalunya uoc.edu

FICHA DEL TRABAJO FINAL

Title:	*InfoHound tool – Improving OSINT open source CyberArsenal for good*
Author's name:	*Xavier Marrugat Plaza*
Supervisor's name:	*Jordi Guijarro Olivares*
PRA's name:	*Víctor García Font*
Submission date (mm/aaaa):	*03/2023*
Degree or programme:	*Master's Degree in Cybersecurity and Privacy*
Thesis' field:	*Business security*
Language:	*English*
Key words	*OSINT, Google Hacking, Tool development*

Abstract

During the reconnaissance phase, an attacker searches for any information about his target to create a profile that will later help him to identify possible vulnerabilities or misconfigurations to exploit. Using passive analysis methods (which do not interact directly with the target) through OSINT, a large amount of data can be extracted, such as: organisation addresses, IPs, systems open ports, sensitive files or directories exposed to the internet and subdomains, among others. Are we aware of this data? Do we know what impact they can have in an organization? What information is publicly available about us?

Currently, there are specific Open Source OSINT tools for each of the data that we want to monitor. This involves setting up and running these programs separately and then grouping the results together. Although there are tools that combine different functionalities to extract more data, such as ReconFTW (an Open Source tool that finds IPs, ports, subdomains and more from the target) and LookingGlass (a private solution that monitors a large amount of internet sources to detect possible threats to an organisation), no Open Source tool has been found that groups a great variety of these techniques together.

Therefore, the aim of this project is to implement an Open Source tool that collects all the information that can be extracted from publicly accessible sources from an Internet web domain. In addition, considering the information obtained, it will create a profile of the company and its employees helping analysts in their cybersecurity assessments.

Index

Figures list

1. Introduction

Nowadays, it is well known that information is one of the key elements for any business. Whether they use their customer's data to maximise their profits, discover new market opportunities or both; information is power. However, they do also generate data that can be interesting for other people, attackers.

1.1. Context and justification of the work

Every time something is published on the Internet it gets indexed by many services. For example, Google indexes web pages **[1]** to be able to list them in case a user searches for a particular topic. The Wayback Machine **[2]**, an Internet archive project, in the other hand, makes copies or snapshots of the state of any accessible website to preserve the history of the Internet. There are many other services that indexes the content of the internet for many other reasons. The important aspect is that, whenever something gets indexed, it is very difficult to track where the information is and remove it.

Open Source Intelligence (OSINT) **[3]** refers to the knowledge acquired from publicly accessible sources. A lot of tools have been developed to use OSINT techniques to retrieve sensitive information that can be later used by cybersecurity analysts or attackers. Those that are Open Source usually are just capable to search for one type of data. Unfortunately, this causes analysts to expend a lot of time executing each tool and parsing the results to use them in another one. Although it exists some of them that combine different tools, they usually lack some types of data and it is very difficult for analysts to develop their own code as a tool extension.

In this project, a multi-modular tool (named InfoHound) will be developed. Although implementing the ability to retrieve and analyse every type of data from every data source is not feasible, InfoHound will be designed to accept custom modules developed by users so that they can adapt the tool to their special needs. Also, it will let users visualize all data collected through a web application to facilitate the analysts' work.

1.2. Objectives

The objectives of this master thesis are described in the following list:

- Raise awareness about the data that can be retrieved using publicly available sources.

- Show how attackers use OSINT tools to gather any possible type of information from their target that can help them later prepare an attack.

- Show how useful and powerful Google Hacking **[4]** (or dorking) techniques can be to get information from a target.

- Develop a multi-modular python tool that, given an internet domain name, retrieves and analyses information from open data sources in order to identify sensitive information and potential vulnerabilities.

- Design and implement the modular python tool so it can accept custom modules made by other analysts so they can adapt it to their needs.

- Design and implement a web application in order to easily visualize the data collected.

1.3. Sustainability, social-ethical and diversity impacts

The MITRE ATT&CK matrix **[5]** is one of the main frameworks that defines adversary tactics and techniques based on real-world observation. As we can see in it, the first step in any attack is the reconnaissance. Then, it is wisely to consider that any malicious actor will perform at least one of those techniques to gather as much information as possible from their target.

As explained in the previous section, one of the objectives of this master thesis is to developed a tool that will let anyone gather data from a target. Although this can be used by attackers for their benefit, the main goal is to simplify this task for those analysts that want to know which information their company has exposed on the internet and apply measures to protect it from malicious actors.

Moreover, all the knowledge gathered from the state of the art and the emphasis done in this document about the risk of exposing certain type of data on the internet, can be used to create awareness through all people, not only IT.

For these reasons, it is foreseen that this master thesis will have a great impact on society. Helping business and people know what internet has about them. However, it will not help much in the sustainability of the planet as it does not try to neither reduce CO2 emissions nor create affordable clean energy, among other things.

1.4. State of the art

The act of discovering sensitive data from a target in order to use it against it, it is not something new. In fact, it has been done all over the history to know the weaknesses of an adversary to exploit them.

Today, in the era of information, there is no need to send people all over the globe to spy on a target. You can find almost any information you want on the internet. Luckily, for cybersecurity analysts, this task can be easily done by the use of one of the many automated tools that people have developed.

To know the capabilities of these type of tools, the following list shows in detail the most relevant ones. Those which only gather one type of data (e.g. emails) are at the beginning as the multi-purpose ones, that collects all kind of information, usually combine many of them:

- Pagodo **[6]**: this tool lets the user perform Google Hacking automation. The aim of this tool is to search for applications vulnerabilities and sensitive data exposed directly from Google (or other browsers) by the use of dorks (advanced queries). To do so, it uses some techniques to evade Google captcha as the preferred way is by using Google's API which it is not free.

- Hunter **[7]**: this service allows users to search emails related to a given domain or an enterprise name. It has a paid API but it will show some emails for free.

- Shodan **[8]**: this web application let us find the ports a server has open without having to perform any active scanning by our side. Also, it can detect misconfigurations and vulnerable services on the target. However, this tool is not free.

- FOCA **[9]**: is an open source tool that finds hidden information in documents. Although it does not fall into an OSINT category tool, it can be used to discover emails, usernames, software versions and operating systems.

- h8mail **[10]**: this tool uses different paid services to get passwords given an email.

- Amass **[11]**: this project from OWASP **[12]** performs a network mapping of attack surfaces and try to discover external assets through the use of open source services and active reconnaissance techniques. Among the data that can be retrieved using these techniques we find: emails (Hunter), passwords (IntelX **[13]**), whois data, web snapshots (Wayback), open ports (Shodan), etc. It is worth noting that, to be able to use some of the services, it requires paid API keys.

- ReconFTW **[14]**: this tool is the most used and popular solution to automate the entire process of reconnaissance. It is actively developed and maintain by the community and has a lot of features implemented using of the tools mentioned above. It has one particular downside, some of the techniques used are very basic (for example the Google Dorks part) and it does not allow custom user-made modules.

- Maltego **[15]**: the aim of this tool is to help analysts easily visualize all the data that has been collected through the use of OSINT data sources available in their marketplace. Almost all of these sources require a paid API key. One good thing Maltego has is the ability to develop your own module.

Although these are only the most used tools, there are more Open Source ones on the Internet that let people gather almost any kind of information (e.g. phone numbers, names, emails, usernames, passwords…).

However, as we have seen from the list, it has not been found a multi-purpose tool that really makes use of two main topics: browser hacking (dorks) and metadata analysis. Also, almost all the reviewed tools do not directly support custom modules developed by external people making it more time-consuming for analysts who have to jump from different tools to get the same amount of data.

The proposed solution is a multi-modular tool that has some modules to retrieve and analyse information from publicly available data sources and will let the community (other users that use the tool) to develop their own ones to satisfy their needs. This will also allow to have all the information centralized in one place making it easier when doing a cybersecurity assessment.

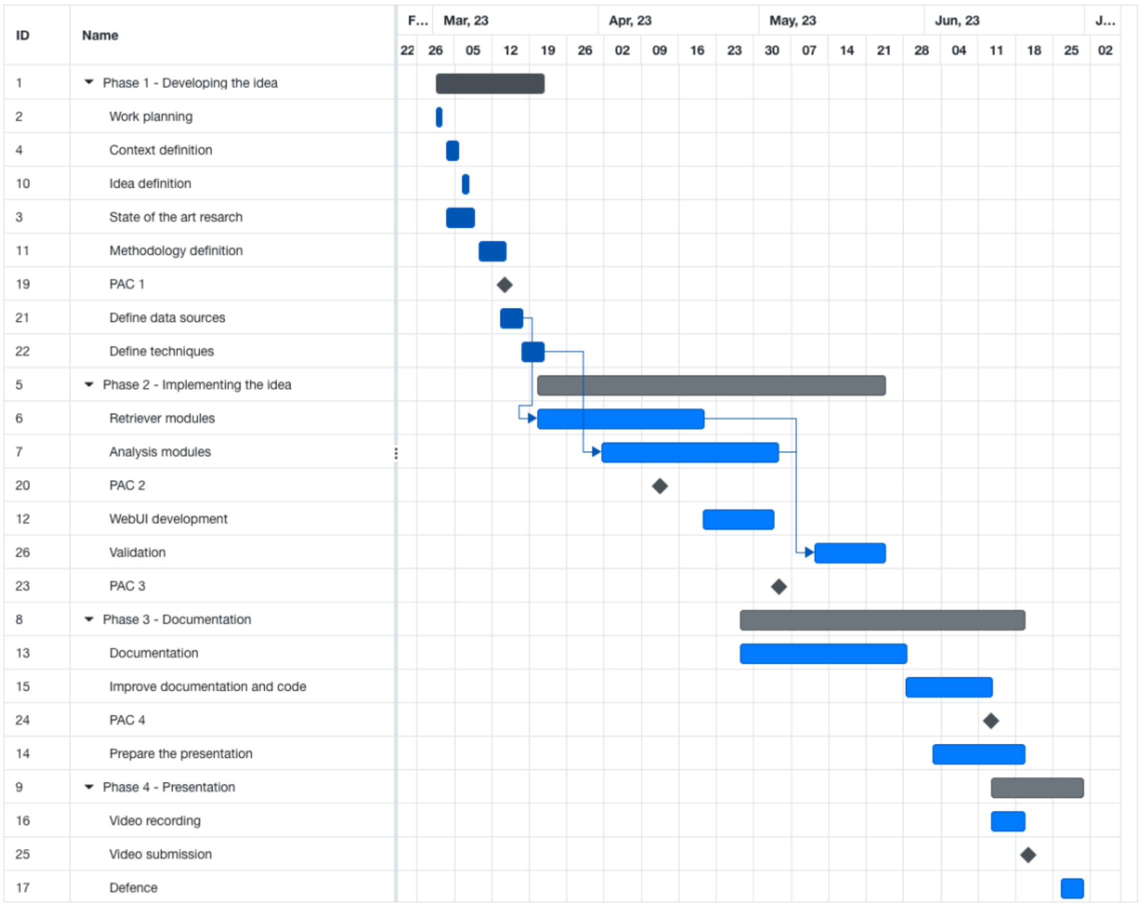
1.5. Work planning

To correctly develop this project, a work planning has been defined. The following list contains the detailed tasks for each of the project phases:

- **PHASE 1** – Developing the idea
  - Work planning: separate the project in tasks and create a Gantt graph to easily visualize the project workload.
  - Context definition: explain the context of the project and where does it fit in the real world.
  - Idea definition: define the idea and its main goals.
  - State of the art research: see which tools and techniques are currently used by attackers and cybersecurity analysts and find possible ways to improve them.
  - Methodology definition: explain which methodology will be used to identify, implement and verify possible solutions for the project. This section also explains how the tool is designed.
  - Define data sources: search which open data sources the project should use in order to retrieve information.
  - Define techniques: check which techniques should be implemented in the project so it has some differentiative aspects among the currently most popular ones.
- **PHASE 2** – Implementing the idea
  - Retriever modules: implementation and definition of the modules in charge of finding all type of data.

- <u>Analysis modules</u>: implementation and definition of the modules in charge of analyse the found data to extract more information.
- <u>WebUI development</u>: implementation and definition of how the web application is developed and its functionalities.
- <u>Validation</u>: check if the tool really analysts in their tasks and see if it improves some aspects over other solutions

- **PHASE 3** – Documentation
  - <u>Documentation</u>: Writing this document.
  - <u>Prepare the presentation</u>: search and user resources to create the slides. Also, some time has been saved to practice for the presentation.
  - <u>Improve documentation and code</u>: take a look at this document and see ways to improve it and make the code clearer so it can be understood by other users.
- **PHASE 4** – Presentation
  - Video recording
  - Thesis defence



ID	Name
1	Phase 1 - Developing the idea
2	Work planning
4	Context definition
10	Idea definition
3	State of the art resarch
11	Methodology definition
19	PAC 1
21	Define data sources
22	Define techniques
5	Phase 2 - Implementing the idea
6	Retriever modules
7	Analysis modules
20	PAC 2
12	WebUI development
26	Validation
23	PAC 3
8	Phase 3 - Documentation
13	Documentation
15	Improve documentation and code
24	PAC 4
14	Prepare the presentation
9	Phase 4 - Presentation
16	Video recording
25	Video submission
17	Defence

1.6. Brief summary of products obtained

At the end of this master thesis there will be available a python tool that, given an internet domain name, it will retrieve and analyse public information related to it without being noticed by the target. It will also output all the sensitive and potentially vulnerable data found in the form of a web application so it can be checked during a cybersecurity assessment.

2. Materials and methods

In order to develop this master thesis, the OSINT cycle has been used as a reference to define the following methodology [16].

Firstly, it will be necessary to perform some research on what type of data is relevant during a cybersecurity assessment so that the resulting tool can satisfy those needs.

Secondly, looking for tools that are being used now by analysts will help identify how they retrieve such information and where they get it from by analysing their code. This is important because we will be able to discover and use several open data sources for each type of data, so the developed tool can rely on several of them instead of just one.

Once we have the type of data and we know where to collect it from, we will need to define how to process it so it can be analysed more easily later.

At this point, we will be ready to start implementing the tool. To do so, Python has been chosen as it is one of the easiest programming languages and very popular among cybersecurity related tools too. In fact, Python will be used too to develop a web application that will be responsible of showing all the collected and processed information so it can help visualize the current state of an investigation. Also, it will help us discover other types of data that could help analysts and the process will start again from the beginning.

This procedure described above will be the main workflow used to implement data types. In an initial research, the main data types detected to be useful are:

- Email addresses
- Phone numbers
- Names
- Usernames and passwords
- Google Hacking results
- Subdomains
- Files

Regarding the open data sources, the tool will use the following ones in its initial phase:

- Google
- Bing
- Wayback Machine
- Alienvault OTX

In order to validate and check for false positive results, the tool will be run to search information of a known organization (200 employees) to see how it performs. This will also help us check if it really speeds up the process comparing it to other tools.

2.1. Potential risks

During the development of this thesis there can appear the following risks:

- **[LOW]** As this is the first “big” OSINT tool that I will develop it can appear other problems that will make me go slower on the planning.
- **[LOW]** Being unable to correctly identify which features are the ones that will increase the value of the final tool.
- **[LOW]** Because some of the techniques that will be implemented on the tool requires time to gather the data of a target it can suppose a drawback for analysts that need their results rapidly.
- **[MEDIUM]** Lack of time to implement all data types and their corresponding retriever and analysis modules.
- **[MEDIUM]** Lack of time to implement the web application that will help visualise all the information gathered by the developed modules.

Although it has been identified the above risks, only the lack of time can really affect the project. For this reason, following the methodology described should help identify which features are the more important to develop earlier so the tool can be useful even if we run out of time.

3. Technical details

As mentioned, developing a Python tool is the primary goal of this thesis. In this section, its design and features will be discussed in depth.

3.1. Why Python?

Python is one of the most popular programming languages thanks to its ease of use, simplicity and flexibility **[17]**. It is vastly used as a scripting programming language and, in fact, many exploits are developed in Python.

Because it has a big community behind it is very easy to find all sorts of third-party libraries that can be easily integrated into any project. The cross-platform compatibility it offers is also key in order to develop a tool meant to be used in any operating system.

In conclusion, Python seems to be suitable for this project based on its features.

3.2. What will Infohound do?

Infohound will retrieve all publicly available information given a domain name. The obtained results should provide analysts with an overview of what data malicious actors can gather from, for example, a company. Some of this information will be then analysed to extract additional details or to find vulnerable services. However, the main goal is to help visualize which degree of exposure a web domain has on the internet.

3.3. Infohound tool design

The tool aims to be simple to use and customizable for every need an analyst can have when performing passive enumeration. For this reason, its functionalities have been divided as granularly as possible to create a multi-modular design.

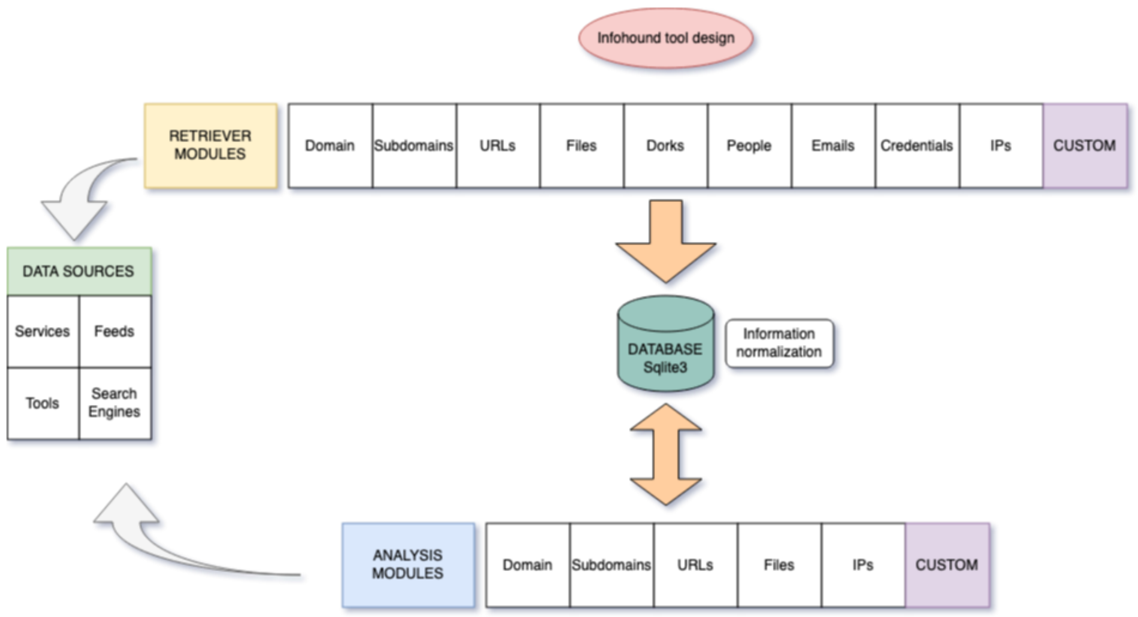


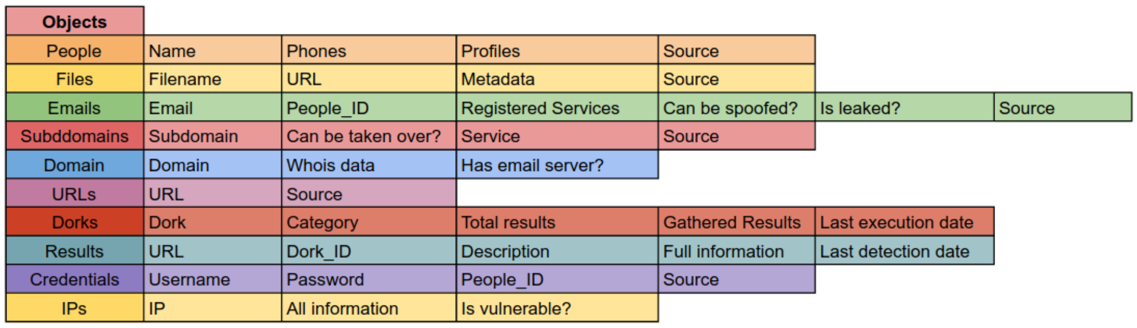
Figure 1 - Infohound design

As can be seen in **Figure 1**, Infohound is divided in 5 big parts: database (containing data types), retriever modules, analysis modules, custom modules and data sources. Each of these groups is reviewed in detail in the following subsections.

<u>**Note**</u>**:** small changes will be made through the development of the tool as new and better ways to divide each task will be found. This will be taken as a baseline to start building the whole tool.

3.3.1. Data types

As mentioned in **Section 2**, after some initial analysis of what kind of data the most popular OSINT tools look for, these have been stablished as the most basic ones: domain information, subdomains, interesting URLs, files, Dorking results, people, emails, credentials and IPs. For this reason, each data type will have its own table in the database with the following columns:



Objects						
People	Name	Phones	Profiles	Source		
Files	Filename	URL	Metadata	Source		
Emails	Email	People_ID	Registered Services	Can be spoofed?	Is leaked?	Source
Subddomains	Subdomain	Can be taken over?	Service	Source		
Domain	Domain	Whois data	Has email server?			
URLs	URL	Source				
Dorks	Dork	Category	Total results	Gathered Results	Last execution date	
Results	URL	Dork_ID	Description	Full information	Last detection date	
Credentials	Username	Password	People_ID	Source		
IPs	IP	All information	Is vulnerable?			

Figure 2 - Data types tables and column names

- <u>Domain</u>: this table will have basic domain data like the actual domain being analysed, all the WHOIS information and whether it has an email server. The

WHOIS field will have all found registries in order to analyse them later to find emails, users, phones or any useful information.

- Subdomains: knowing all the subdomains (active and inactive) allows analysts to monitor its activity and understand how a company operates. They can also be later analysed to see if they can be taken over. This table will store the subdomains found, whether the subdomain can be taken over, which service it is using and in which data source it was discovered.

- URLs: every URL discovered at any stage will be added to this table. This will let other modules use them to find the information they are looking for. For example, it could be that when a file URL is discovered, it is, in fact, pointing to a subdomain so it will be added to this table in order the corresponding module can perform additional checks. This table will also contain in which data source the URLs were discovered.

- Files: this table will contain the URL where the file is located, the name and the whole metadata of each file the tool discovers. All this is important to further extract information about the internals of a company. Metadata can contain a wide range of data. However, each type of file can have different parameters defined. For this reason, instead of redesigning the table each time we found a new one, we store the entire metadata in its column. By doing so, we will be able to revisit all its information whenever it is necessary. Also, at the end, the tool should focus to reduce all the information to one of the basic data types.

- Dorks and Results tables: in these two tables, Dorking related information will be saved. Dorks table will contain the queries to perform, the number of total results Google says to have, the actual results found and when was the dork executed last time. This will help both monitor if the results are complete and create some rules to re-execute them in order to find other vulnerable paths. Then, in the Results table, there will be an entry for every URL found followed by the dork that discovered it, a description of the webpage content, all the information Google returns and when it was seen. As before, the data is stored as raw as possible so it is easy to revisit it in the future by custom modules, for example.

- People: this table will be one of the most interesting for analysist as it will contain names, phones and a list of social media profiles. It will also have where they were found.

- Emails: this table is the most complete. For each entry, it will contain the email, the person (if known) from the People table that belongs to, the services where the email has been used in order to create an account, whether it can be spoofed and if it has been found in a leak. Also, as other tables, it will be shown where the email was discovered.

- Credentials: the goal of this table is to create a database of username and password pairs in order to help analysts in the next steps of a red team exercise. It will also contain (if known) who its owner is and which data source was used in order to find them.

- IPs: this table will contain each IP found, related to the domain that is being analysed, and whether it has some service found vulnerable.

As these tables will be difficult to customize by users (i.e. they will need to change not only the table schema but every function where data is inserted into the database to avoid conflicts), they should create their own. In addition, to avoid redundancy, a foreign key should be used to reference one (or more) of the main data type described above.

3.3.2. Retriever modules

These modules have the mission to request every data source needed to gather and save the information for which they were created. Here Infohound will be defined, more or less, with one retriever module for each data type. The following list shows the main functionalities each default Infohound module will have:

- Domain (domain.py): it will have to retrieve from data sources any information related to WHOIS. Also, it should check whether the domain has a mail service.

- Subdomains (subdomain.py): this module will aim to retrieve subdomains from data sources and from other database tables like URLs.

- URLs (urls.py): this file will retrieve cached URLs from data sources like Web Archive **[2]** where a large amount of these entities are stored. It will look for URLs in other database tables too.

- Files (files.py): it will be responsible to populate Files database table from the URLs found and based on a list of extensions. Following FOCA's methodology **[9]**, these file extensions were selected as can support a lot of interesting metadata information: doc, docx, ppt, pptx, pps, ppsx, xls, xlsx, odt, ods, odg, odp, sxw, sxc, sxi, pdf, wpd, svg, indd, rdp, ica, zip and rar.

- Dorks (dorks.py): this module will load all dorks provided by the user through a file into the Dorks database table. It will also update and create Dorks and Results database table whenever a dork is executed as a query in any search engine.

- People (people.py): for now, it will only be responsible to find social media profiles using an email from Emails table and saving all the information found inside People table. Also, if usernames are found they will be added to the Credentials database table.

- Emails (emails.py): this module will find emails using search engines (each of them will need to be present into the data sources folder) and in other tables such as the metadata column from Files table.

- Credentials (credentials.py): it will be responsible to find credentials from different sources based on the usernames found in the Credentials database table. Remember that here usernames can be both nicknames or emails.

- IP (ips.py): this module will find the IPs the domain is using to host all of its services.

3.3.3. Analysis modules

These modules should not retrieve any data from public sources, instead, they should use already known information to process it and extract additional one. However, this distinction can sometimes be difficult to identify and some analysis modules will need to perform checks against some tools or services. As a reference, the following analysis modules have been identified useful:

- Domain (domain_analysis.py): this analysis module will perform some analysis on the WHOIS information found in its retriever module.

- Subdomain (subdomain_analysis.py): this module will be responsible to perform additional checks like, for example, detect the service behind a subdomain and check if can be taken over.

- Email (email_analysis.py): this module will detect the services where an email has been registered to and check whether the email domain can be spoofed.

- Files (files_analysis.py): it will handle the process to download each file and the metadata extraction.

- IPs (ips_analysis.py): this module will perform the necessary checks to determine if a server hosts a vulnerable service or sensitive information.

As can be seen, not all basic data types have for its corresponding analysis module. This is because at the moment of writing it has not been found any further analysis that can be performed on URLs, dorks, people and credentials modules. However, it may appear the need of them during the development of the tool.

3.3.4. Custom modules

In order to let users, add their own modules (retrieval and analysis), a special directory will be created to place them there. These modules will be later imported by the main script and executed.

Although each module created in this thesis has several functions, custom ones should do only a specific task to maintain modularity and granularity.

To correctly create a custom module, the user will have to implement a function named "execute" which will be called by the main program. This mandatory function should do all the actions the user wants to be performed by the module.

For now, this is how a custom module should be created, but it may later be necessary to create an optional "whenFinished" function which will allow the user to stablish what should be done when his module finishes its execution.

3.3.5. Data Sources

Different types of data sources will be used:

- Search engines: two big search engines have been selected, Bing and Google, to find as much information as possible from querying them. Although they will share some functionalities, Google will be the only one performing Dorking queries for simplicity. At least, at the beginning of the Infohound development.

- Tools: Holehe **[18]**, with more than 120 sites to check, will be used to gather which services an email has been registered to.

- Feeds: at this point of the project only Alienvault OTX (Open Threat Exchange) **[19]** has been identified as useful. It's a free thread hunting tool with an API which allows users to retrieve information about domains. This will mainly be used to find URLs, subdomains, files and WHOIS information.

- Other services: here we can find Web Archive **[2]**, Shodan **[8]** and other tools related to credentials discovery like Leak Lookup **[20]** or Have I Been Pwned **[21]**. At first, Web Archive will help find URLs, subdomains and files that will be possible to download without reaching the domain server. Secondly, Shodan will be used to find more information about the IPs found like ports, services and whether they are vulnerable to some exploits. The credential related services will have to be analysed in depth to check which are the most useful and less expensive.

3.3.6. Possible improvements

When defining what exactly Infohound should do and its design, some flaws were detected. Although they do not directly impact on the results of the tool, they can make a big difference in its performance. However, some of these improvements can be time consuming when developing. To avoid running out of time, they will be considered and implemented while creating Infohound.

- Concurrency problem: performing some tasks can take a lot of time. For instance, when files have to be downloaded in order to extract the metadata, depending on how many files have been retrieved it can last for hours. Implementing the ability to execute asynchronous tasks will reduce waiting times drastically.

- Improving the database: Sqlite3 was considered in this initial design, however, it does not scale efficiently and it neither supports concurrent operations. PostgreSQL **[22]**, in the other hand, allows this type of queries and it performs better. The only small drawback is that it requires to be run as a separate service.

- Django models: if the idea is to create a web interface at the end of the project it should be considered the use of the Python web framework, Django **[23]**. Using its system models to create the tables will give a layer of abstraction and will allow users to decide which database they want to use without modifying anything. Also, the code will be cleaner and it will be easier to understand when the information is modified. This improvement can be easy to make, but it can also be an overkill solution if Django is only used to show static data straight from the database.

- Add more analysis modules: as mentioned above, the analysis modules have to be improved. However, depending on which information wants to be extracted, it can increase its complexity a lot which can lead to running out of time.

4. Technical development

4.1. First iteration – Simple PoC

This iteration has several goals. First, to check the viability of the whole project. Although, we have seen that there are a lot of open source tools doing similar tasks as proposed in this project, it could be possible to find some difficulties when trying to implement this type of tools. Second, this iteration while help us to see if the previously defined database schema needs to be changed as some other important data may be discovered. Third, to detect which are the most valuable information to be gathered and methods to accomplish this task.

This section has been divided in three subsections: retrieval modules, analysis modules and custom modules. In each of them, it will be discussed the first implemented functions.

4.1.1. Data sources

From now, when any external service is used, it will be handled by its corresponding data source. This is done to avoid duplicate code and to facilitate its maintenance. In this first iteration the following data sources are implemented:

- alienvault.py
- archive.py
- bing_data.py
- google_data.py
- leak_lookup.py
- adobe.py
- duolingo.py
- imgur.py
- mewe.py
- parler.py
- rumble.py
- snapchat.py
- twitter.py
- wordpress.py

4.1.2. Retrieval modules

domains.py

- get_whois_info: given a domain name it requests all the data associated to it from the whois registers. The tool used to perform this task is the python whois library **[**24]. Although, it exists other sources to find these records, for example the Alienvault OTX API, it has been observed that they all have the same level of information.

subdomains.py

- get_subdomains: this function will use the Alievault OTX API to retrieve all the domains from the */indicators/domain/<domain>/passive_dns* endpoint (performed by the data source). This call will return all the hostnames Alienvault have detected from a particular domain. For example, for *uoc.edu* it returns 52 unique subdomains which is a very good starting point.

- get_subdomains_from_urls: as all URLs detected on other modules will be added in the URLs database table, it is wise to check all its entries to check if any other subdomain is found.

urls.py

- get_urls**:** one great source to get URLs is Web Archive (implemented as a data source). As discussed earlier, it creates snapshots of the internet to preserve its history. For this reason, is easy to find a lot of URLs from a single domain included those that are no longer available. For example, querying URLs from the *uoc.edu* domain it returns more than 400k URLs. This will help discover new subdomains, files and even emails. Also, as all this URLs have been cached by Web Archive, files and pages can be downloaded and accessed without being detected by the target domain.

files.py

- get_files_from_urls: this function reads every entry in the URLs database table and when a file is detected (i.e. the url is pointing to a file with one of the following extensions: doc, docx, ppt, pptx, pps, ppsx, xls, xlsx, odt, ods, odg, odp, sxw, sxc, sxi, pdf, wpd, svg, indd, rdp, ica, zip and rar) it adds it to the Files database table. As Web Archive files URLs point to a previsualization window and not the actual cached file, it is necessary to change the link. In those cases, the File database entry is added with the downloadable modified link. However, creating this new URL is not as simple as adding some value, because Web Archive has snapshots of each file we request to get the latest one. This is very time consuming and needs to be improved.

dorks.py

- load_dorks_from_file: given a file with dorks and a category, this function will add them to the Dorks database table. In case <*domain*> tag is detected it will be changed to domain being analysed.

- execute_dorks: this function will query each dork to Google (using its data source) and store its output to the Results database table.

emails.py

- find_emails: this function will use Google and Bing data sources to look for emails related to the domain being analysed. It will do it by using the following dork (using uoc.edu as example): *intext:@uoc.edu* and *inbody:@uoc.edu* respectively.

- find_emails_from_urls: sometimes parameters can be cached too so it is worth checking if an email is being indexed by any engine, especially in Web Archive.

people.py

- find_social_profiles_by_email: given the emails found, this function searches in Google splitting each: from *example@uoc.edu* to the dork *"example" uoc.edu*. Then it will try to find in the results, Twitter and LinkedIn profiles extracting the person name (to add an entry in the People database table) and usernames (to add them in the Credentials database table).

credentials.py and **ips.py**

In this first iteration these two modules were skipped. Firstly, because checking searching for credentials and IPs seemed to be trivial in the form of using an external service or performing a name resolution. And secondly, to focus in other modules that seemed to have more work and difficulties.

4.1.3. Analysis modules

domain_analysis.py

- can_be_spoofed: DMARC policy is a parameter in the email server configuration that will stablish what to do when the sender of an email header and the sender of the email body are not the same (i.e. email spoofing). This function, will check this policy in order to determine if the email domain can be spoofed. Although, this functionality could have been added in the **emails_analysis.py** file, it seemed right to put as a domain check rather than for each email found. To implement this function, SpoofThatEmail [25] tool was taken as a reference.

- subdomain_takeover_analysis: for every subdomain found, this function will check if it can be taken over. Taking over a subdomain [26] can be tricky as there are a lot of services. For this reason, "Can I take over XYZ?" [27] repository has been used as a reference, in fact, only its *fingerprints.json* file that contains fingerprints and other checks for every service.

files_analysis.py

- download_all_files: this function will download each file found in the Files database table to the *downloaded_files* directory. This will allow to perform further analysis on them. As hundreds of files can be found, the time spent to get all of them can be very high. For this reason, this function will need to be improved to be able to download several files at the same time.

- get_metadata: using the ExifTool [28] package, the metadata of each downloaded file will be extracted and saved entirely to the database.

- get_emails_from_metadata: this function will loop over each Files database table entry to get the metadata and extract all emails found.

emails_analysis.py

- find_registered_sites: as emails are used to create accounts over internet platforms, it is worth checking if the emails Infohound found were used too. To perform this task, Poastal [29] tool has been used as a reference as it checks on popular platforms such us Facebook, Twitter, Snapchat, Parler, Rumble, MeWe, Imgur, Adobe, WordPress, and Duolingo. Each of this site will have its data source file.

- check_breach: this function will check if the email is found in a data breach. There are many services that can perform this task however, most of them are limited or require a paid subscription. Leak-lookup [30] has been chosen as it has a public API to perform simple checks (without showing the actual password) although it only allows 10 daily requests. It has been added as a data source.

4.1.4. Custom modules

In this iteration, custom modules were not implemented.

4.2. Second iteration – Django

Within the first iteration it could be seen that the project was really viable and there was room to improve. In this section it will be discussed the upgrades done in each module and in the whole project.

4.2.1. General improvements

a) Using Django: as said in previous sections, the output of the tool will be visualized in a web page. For this reason, and sticking to Python language, Django can easily perform this task. Also, this will allow us to forget about future problems when changing the database technology as Django stablishes its model schema.

b) PostgreSQL: in the first iteration we detected that a lot of records can be gathered for several database tables. To handle that amount of data efficiently, it was necessary to migrate from SQLite3 to PostgreSQL. Using Django this task was as easy as changing the configuration of it rather than finding and modifying each SQL query in the whole tool.

c) Concurrency: we saw that there are some functionalities that can take a lot of time to complete as they need to perform some kind of processing to request to other services. Trio is a Python library for asynchronous concurrency and I/O, and it has been implemented to solve this problem.

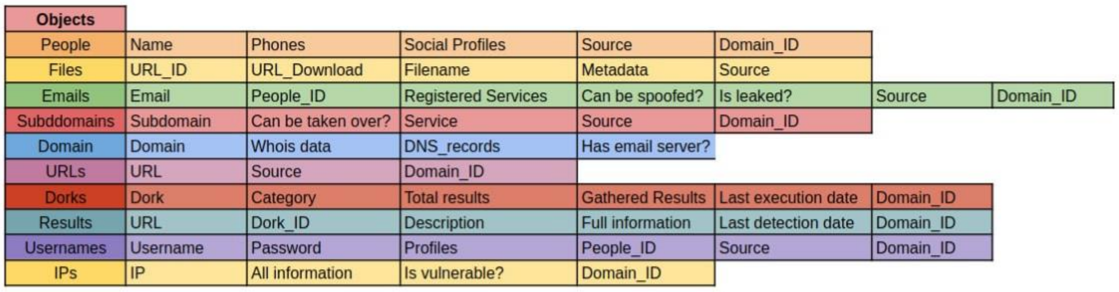
d) Tables properties: changes were made in some of the tables:

- In the Peoples model (the table equivalent in Django) the “Profiles” property has been renamed to “Social Profiles” because it will only contain LinkedIn and Twitter links pointing to the person profile.

- A new property has been added into the Domain model: DNS_Records. This will store the A, AAAA, MX, NS and TXT records.

- The Credentials model has been renamed to Usernames as it better represents the information stored in there. Now, it will also have a “Profiles” property which will contain a list of profile links to services where that username is being used.

- A new property has been added to every model, the “Domain_ID”. This will allow the Infohound tool to handle several domain analyses at the same time.



Objects							
People	Name	Phones	Social Profiles	Source	Domain_ID		
Files	URL_ID	URL_Download	Filename	Metadata	Source		
Emails	Email	People_ID	Registered Services	Can be spoofed?	Is leaked?	Source	Domain_ID
Subddomains	Subdomain	Can be taken over?	Service	Source	Domain_ID		
Domain	Domain	Whois data	DNS_records	Has email server?			
URLs	URL	Source	Domain_ID				
Dorks	Dork	Category	Total results	Gathered Results	Last execution date	Domain_ID	
Results	URL	Dork_ID	Description	Full information	Last detection date	Domain_ID	
Usernames	Username	Password	Profiles	People_ID	Source	Domain_ID	
IPs	IP	All information	Is vulnerable?	Domain_ID			

Figure 3 - Improved database properties

4.2.2. Data sources

Some changes and parsing improvements have been made. Also, a new file structure has been defined, later discussed.

4.2.3. Retrieval modules

domains.py

- get_dns_records**:** this function will look for A, AAAA, MX, NS and TXT DNS records from a domain. This information can be useful to, for example, determine which email service is being used and to find other information

subdomains.py

Nothing has been added or changed in this iteration.

urls.py

Nothing has been added or changed in this iteration.

files.py

- get_files_from_urls: this function has been divided in two. Instead of handling all the process to retrieve the file download URL from Web Archive, which is time consuming, now it retrieves the entries of the URLs model and passes the link to another function to perform the heavy work. This allows a better integration with Trio library to create concurrent calls which greatly improves its performance.

dorks.py

Nothing has been added or changed in this iteration.

people.py

Nothing has been added or changed in this iteration.

emails.py

Some improvements have been made. Now all found emails are being lowercased and validated before adding them into the database to prevent false positives and duplicated entries.

usernames.py

- find_usernames_from_people: this function adds the usernames found in Twitter or LinkedIn to Usernames model. This task was done earlier by people.py when looking for social profiles given an email.

- find_usernames_from_emails: some services use the email as a username, that is why the emails found will be used too to find possible matches in several platforms.

- It is being studied to add a function to generate possible usernames based on people's name performing some type of variations. Although this can increase our finding it will also output a lot of false positives.

ips.py

This module has only one function, given an IP it gets all its related information available in Shodan to be later analysed. The main drawback is that to use Shodan's API the user will need to have a paid subscription as it only allows a few queries per month.

4.2.4. Analysis modules

domain_analysis.py

Nothing has been added or changed in this iteration.

files_analysis.py

- download_all_files: in order to improve the performance of this function, it has been used the Trio library to make concurrency calls to another function that performs the actual file downloading process.

- get_emails_from_file_content: reading the content of several file types can be tricky. At this time, only text is extracted to be later analysed from file types: doc, docx, ppt, pptx, pps, ppsx, xls, xlsx, pdf, svg, indd, rdp, ica and rar.

email_analysis.py

Nothing has been added or changed in this iteration.

usernames_analysis.py (new)

- get_profiles: Given the usernames found, this function will pass them to Maigret [32] tool which is a fork of Sherlock [33]. These two tools check usernames on a huge number of sites. More specifically, Maigret currently supports more than 3000 sites without requiring an API key.

4.2.5. Custom modules

In this iteration, the support of custom modules has been implemented. It works very simple. The only thing the user who wants to create a custom module must do is to create a Python file inside *custom_modules* directory and implement a function named *custom_task.* The file can contain other functions that gives support to this main one.

Infohound will import all files it finds in this directory and execute the *custom_task* function. As an example, it has been added the tool Holehe (previously discussed) which given an email address checks if it has been used to register to specific services. Holehe supports more than 120 sites although some of them stablish a request rate limit and can give false negatives. The way this tool is added as a custom module is as specified before:

1.1 Create a file named *holehe.py* inside *custom_modules* directory.

2.1 Import necessary packages, including *holehe*, Emails model and Trio.

3.1 Create the function *custom_task*. This function will immediately call another one present in the same file, *find_registered_sites_holehe.* As can take some time to gather all the information, this function has been implemented as an asynchronous one.

4.2.6. Results in numbers

The test following results have been obtained analysing a company which has more than 150 employees.

- Number of URLs: 22191
- Number of subdomains: 103
- People: 13
    - Most of them with Twitter or LinkedIn
    - Four of them with two usernames
- Emails: 18
- Files: 726

It is important to note that when a function is executed, it can add additional information useful for a previous executed function. So, although this numbers might seem low not all functions have been run again. At this stage of the project, the goal has been to implement functionalities rather than trying to perform a deep analysis. Now, that Infohound tool has a great base, one of the next steps is to improve the results and add more capabilities.

4.2.7. File structure

As a lot of files and directories have been discussed, it can be hard to follow the overall project structure. The main folders inside the project are:

- data_sources: will contain the implementations to query services like Google, Bind, Web Archive, AlienVault and other. It also has one folder for leaks services (like LeakLookup) and another one that contain services to check if certain emails have been registered there.
- analysis_modules: has the modules that analyse the data already present in the database.
- retrieval_modules: contains the modules in charge to gather as much information as possible about the target domain.
- custom_modules: will has user created modules.
- dorks: in order to import the dorks to be executed by the tool the user will need to provide them in a text file where one line represents one dork. This file will need to be placed in this folder.

Here you can find a simplified schema:

```
infohound/
├── data_sources/
│   ├── leaks/
│   │   └── leak_lookup.py
│   ├── services/
│   │   ├── adobe.py
│   │   ├── duolingo.py
│   │   ├── imgur.py
│   │   ├── mewe.py
│   │   ├── parler.py
│   │   ├── rumble.py
│   │   ├── snapchat.py
│   │   ├── twitter.py
│   │   ├── wordpress.py
│   │   └── //more services could be added
│   ├── alienvault.py
│   ├── archive.py
│   ├── bing_data.py
│   ├── google_data.py
│   └── shodan.py //New data source
├── analysis_modules/
│   ├── domain_analysis.py
│   ├── email_analysis.py
│   ├── files_analysis.py
│   ├── fingerprints.json
│   └── usernames_analysis.py
├── retrieval_modules/
│   ├── domains.py
│   ├── dorks.py
│   ├── emails.py
│   ├── files.py
│   ├── ips.py
│   ├── people.py
│   ├── subdomains.py
│   ├── urls.py
│   └── usernames.py
├── custom_modules/
│   └── holehe.py
├── dorks/
│   └── dork_list.txt
├── infohound_utils.py //Stores API keys
└── main.py
```

4.3. Third iteration – The web app

With all the progress done in terms of functionalities, the next big step is the creation of the web application. The initial idea was to create a basic html report where the data would be displayed. But creating a web application seemed to be a good idea to give the user the ability to dynamically execute tasks based on the information that previously-executed tasks found.

4.3.1. Improvements from previous iteration

a) Web application: in this iteration a backend and a frontend have been developed. The web app has been designed with the idea to be easy to use and understand all the found data. InfoHound now is a complete tool with a fully functional web.

b) Tasks and progress bars: as the tool have some functionalities that can be executed more than one time (e.g. new email addresses have been found and it is possible to search for related people) it is mandatory that the analyst has the ability to launch some tasks the backend must do. It has been possible to develop an architecture to handle these operations.

c) Improve modules: some modules have been updated to fix minor errors and to improve their features.

d) New data sources: subdomains can now be gathered from CRT.sh and HackerTarget sources.

e) Dorks: a folder is created so the analyst can put there the dorks to be loaded in the application. The category of the dork is determined by the filename. Six files have been added to this folder containing some dorks for the following categories: exposed services, find files, general information, information in external services (GitHub, Confluence, Trello, etc.), possible vulnerabilities and search emails.

f) Table properties: changes were made in some tables:

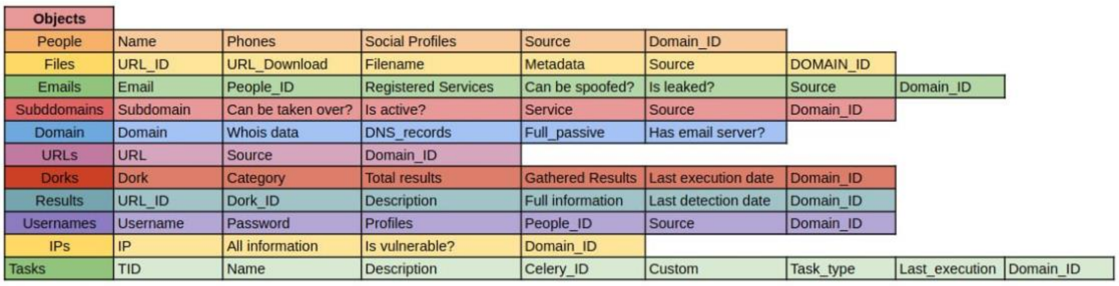
- The Domain table now has a Boolean parameter to determine if the analysis to be performed should be 100% passive. If false, the URLs that point directly to files hosted on the specified domain will be downloaded and analysis. However, if *full_passive* is set to true, those will not be queried as the target could log the activity.

- A new column has been added to Subdomains table where a Boolean will indicate whether the subdomain is active.

- The URL in Results table now is a foreign key pointing to an entry from URLs.

- A new table has been created, Tasks. This will be used to keep track of the available tasks the analyst can perform through the web application and their state.

- The IP table is left in the database but the functionalities have not been implemented due to lack of utility, for now. Although it is true it could give a lot of helpful information using services like Shodan, it has been noticed that a lot of domains are behind hosting or other technologies to mask their real IP making it difficult to correlate the findings to the domain being analysed.



Objects								
People	Name	Phones	Social Profiles	Source	Domain_ID			
Files	URL_ID	URL_Download	Filename	Metadata	Source	DOMAIN_ID		
Emails	Email	People_ID	Registered Services	Can be spoofed?	Is leaked?	Source	Domain_ID	
Subddomains	Subdomain	Can be taken over?	Is active?	Service	Source	Domain_ID		
Domain	Domain	Whois data	DNS_records	Full_passive	Has email server?			
URLs	URL	Source	Domain_ID					
Dorks	Dork	Category	Total results	Gathered Results	Last execution date	Domain_ID		
Results	URL_ID	Dork_ID	Description	Full information	Last detection date	Domain_ID		
Usernames	Username	Password	Profiles	People_ID	Source	Domain_ID		
IPs	IP	All information	Is vulnerable?	Domain_ID				
Tasks	TID	Name	Description	Celery_ID	Custom	Task_type	Last_execution	Domain_ID

Figure 4 Improved database properties II

4.3.2. Web app

The web application has 3 sections:

- Domain list: located at the left, it lists all the domains added by the user. As mentioned previously, being able to analyse multiple domains at the same time can be very helpful to an analyst. It has been designed to make it easy to know what information is being viewed at any given time.

- Navigation bar: there are 5 tabs for each domain. The data displayed in them is described in the next section.

- Content: here the user will see the results gathered from the tool.

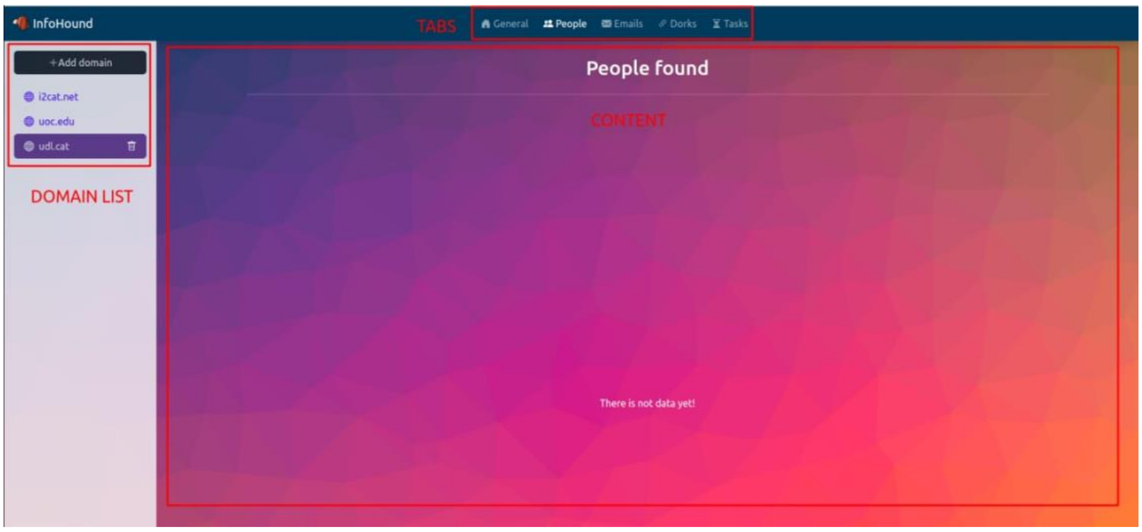


Figure 5 InfoHound Web Application

Adding a domain

Adding a domain is as easy as clicking on the "Add domain" button at top of the domain list. A pop up will appear where the user can enter the domain name and check if the analysis should be fully or partially passive.

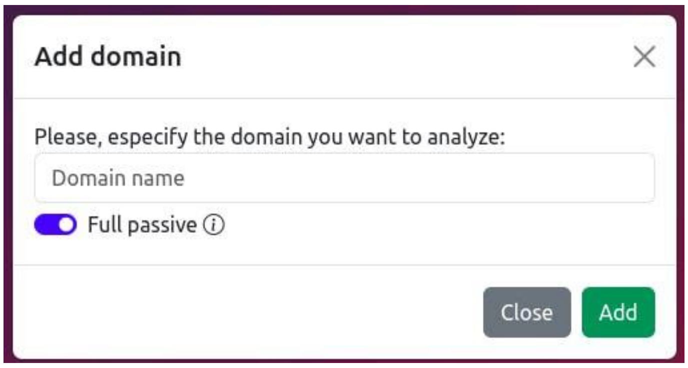
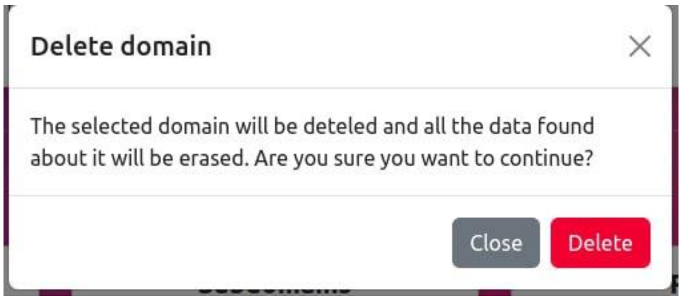


Figure 6 Add domain window

After submitting the form, the domain will be added in the domain list and some initial task (e.g. get DNS records) will be executed.

Deleting a domain

If the domain has to be deleted from the domain list, it is as simple as clicking in the bin icon it appears next to the domain when selected.


Delete domain

The selected domain will be deteled and all the data found about it will be erased. Are you sure you want to continue?

Figure 7 Deleteing a domain

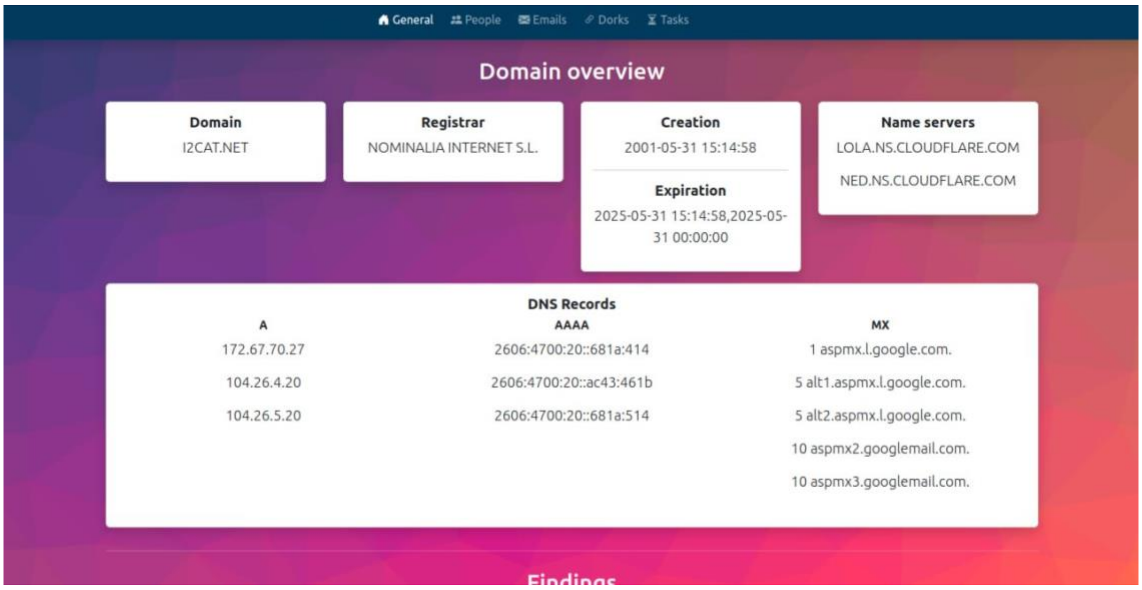
Another pop up will be shown to the user alerting of the deletion of all entries in the database for that specific domain.

4.3.3. Tabs and content

The web application has 5 tabs: general, people, emails, dorks and tasks. In this section it will be discussed which information can be found in each these tabs.

General

This tab gives a general overview about the findings. For instance, it shows the information given from a whois query and the DNS records if found.



Domain overview

Domain	Registrar	Creation	Name servers
I2CAT.NET	NOMINALIA INTERNET S.L.	2001-05-31 15:14:58	LOLA.NS.CLOUDFLARE.COM
		Expiration 2025-05-31 15:14:58,2025-05-31 00:00:00	NED.NS.CLOUDFLARE.COM

DNS Records

A	AAAA	MX
172.67.70.27	2606:4700:20::681a:414	1 aspmx.l.google.com.
104.26.4.20	2606:4700:20::ac43:461b	5 alt1.aspmx.l.google.com.
104.26.5.20	2606:4700:20::681a:514	5 alt2.aspmx.l.google.com.
		10 aspmx2.googlemail.com.
		10 aspmx3.googlemail.com.

Figure 8 General tab - Domain Overview

There are some stats about the general findings of the tool too:

- Total number of URLs gathered

- Total number of subdomains discovered
- People found
- Files found
- Number emails that contains leaks
- Number of spoofable emails
- Number of emails that have been possible to link to a person
- Number of emails that has been used to log in other services

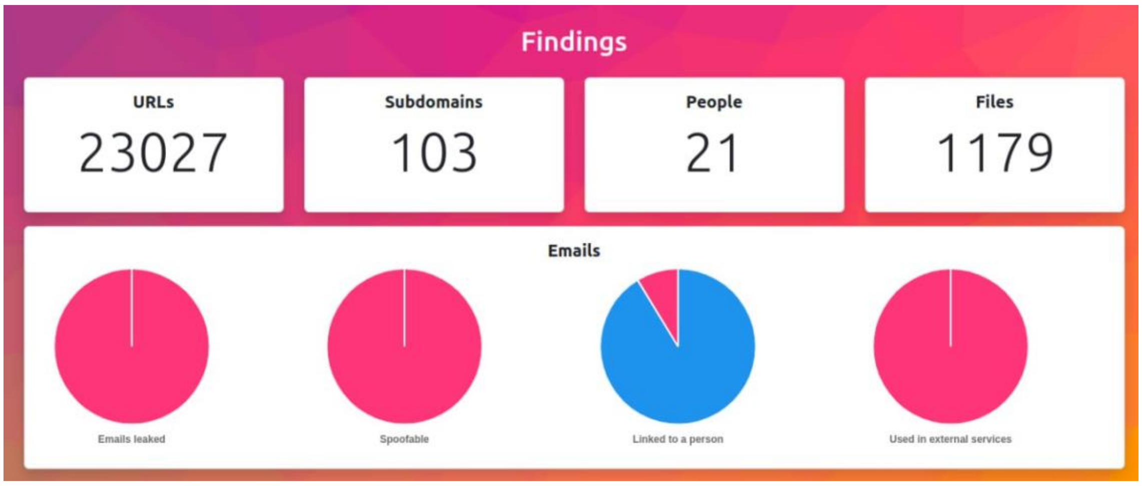
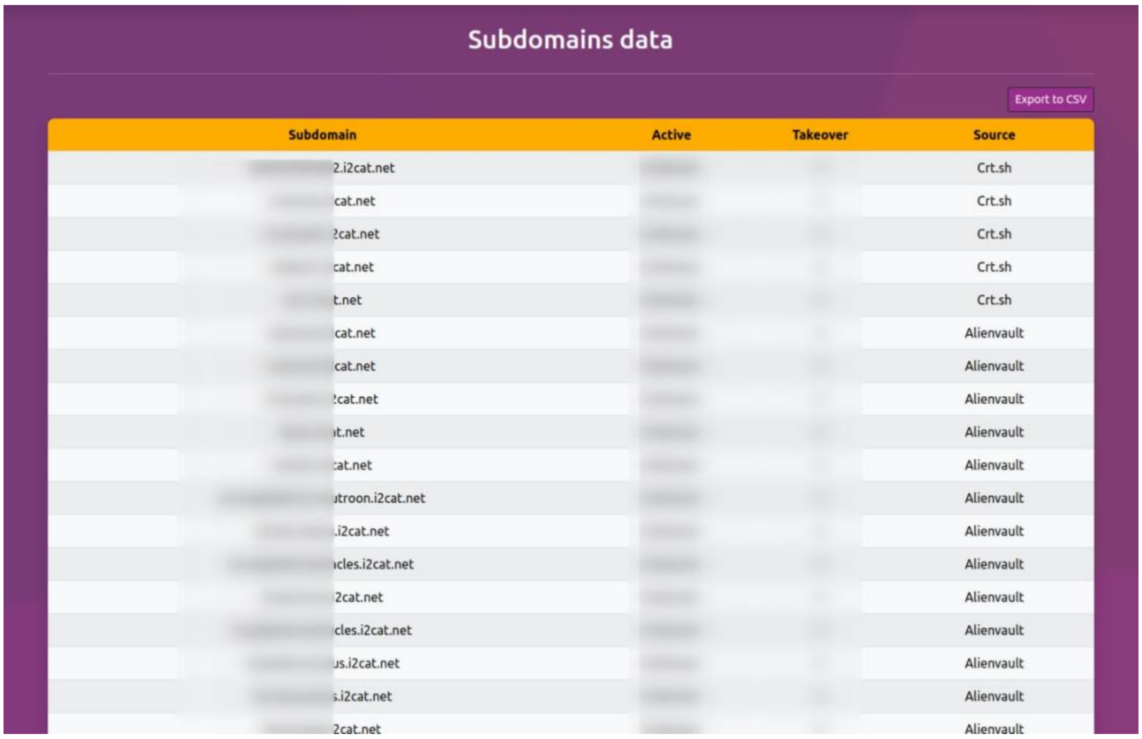


Figure 9 General tab - Findings

Subdomains

This tab shows all the subdomains found, where they have been found and whether they are active or/and can be taken over. Also, there is a button which allows the analyst to export the findings to a CSV file.



Subdomain	Active	Takeover	Source
[redacted]2.i2cat.net	[redacted]	[redacted]	Crt.sh
[redacted]cat.net	[redacted]	[redacted]	Crt.sh
[redacted]2cat.net	[redacted]	[redacted]	Crt.sh
[redacted]cat.net	[redacted]	[redacted]	Crt.sh
[redacted]t.net	[redacted]	[redacted]	Crt.sh
[redacted]cat.net	[redacted]	[redacted]	Alienvault
[redacted]cat.net	[redacted]	[redacted]	Alienvault
[redacted]2cat.net	[redacted]	[redacted]	Alienvault
[redacted]at.net	[redacted]	[redacted]	Alienvault
[redacted]cat.net	[redacted]	[redacted]	Alienvault
[redacted]utroon.i2cat.net	[redacted]	[redacted]	Alienvault
[redacted].i2cat.net	[redacted]	[redacted]	Alienvault
[redacted]acles.i2cat.net	[redacted]	[redacted]	Alienvault
[redacted]2cat.net	[redacted]	[redacted]	Alienvault
[redacted]cles.i2cat.net	[redacted]	[redacted]	Alienvault
[redacted]us.i2cat.net	[redacted]	[redacted]	Alienvault
[redacted]s.i2cat.net	[redacted]	[redacted]	Alienvault
[redacted]2cat.net	[redacted]	[redacted]	Alienvault

Figure 10 Subdomains tab

<u>People</u>

In this tab, the people found is shown in a grid design. It is possible for the user to easily see the emails, phones, usernames/credentials and services or profiles linked to each person the tool has identified. It also shows the icons of Facebook, Twitter and LinkedIn in case the profile has been discovered.

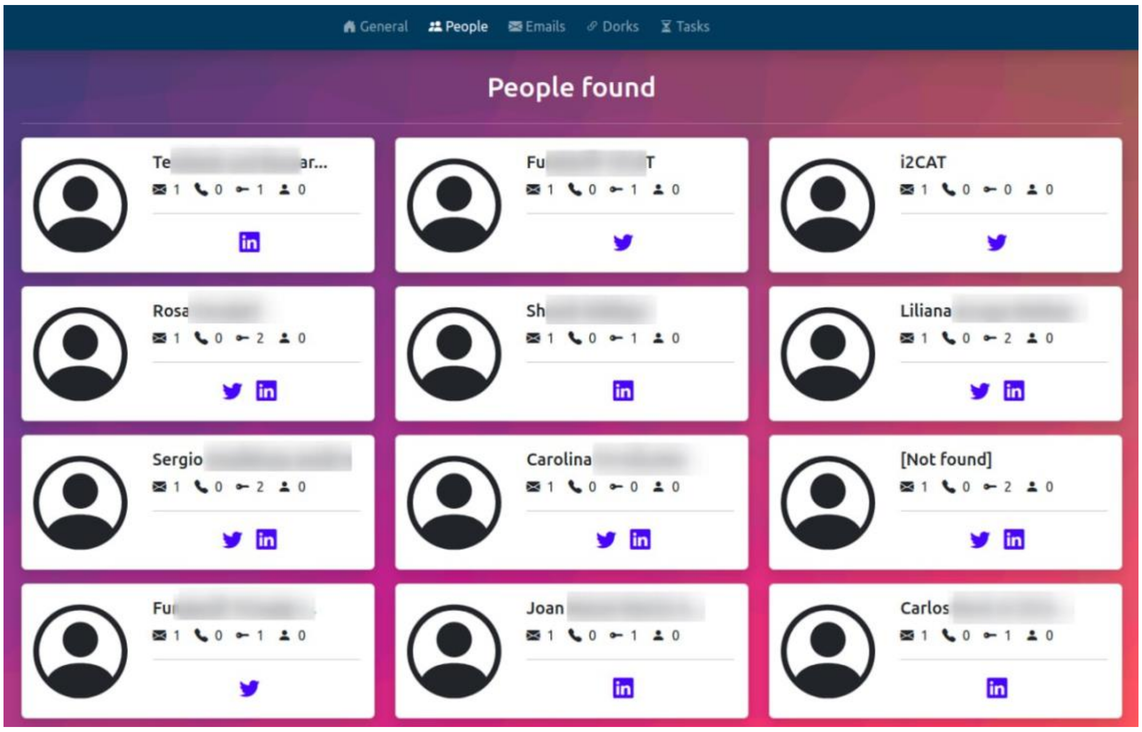


Figure 11 List of people discovered

When clicking any of those people a new window is opened with further information about the person. There we can find several things:

- the email addresses associated to that person,
- the services where email has been found to be registered
- the usernames, whether they appear to have a leak and, in case the analyst has entered the Leak Lookup API key, the password
- the services where it has been found to exists the username

As explained in previous section the services are found using other tools, *Holehe* for emails and *Maigret* for usernames. These two tools can produce false positives results so it has been marked with a tooltip so the analyst is aware of the situation and can understand more precisely what is shown in the application.

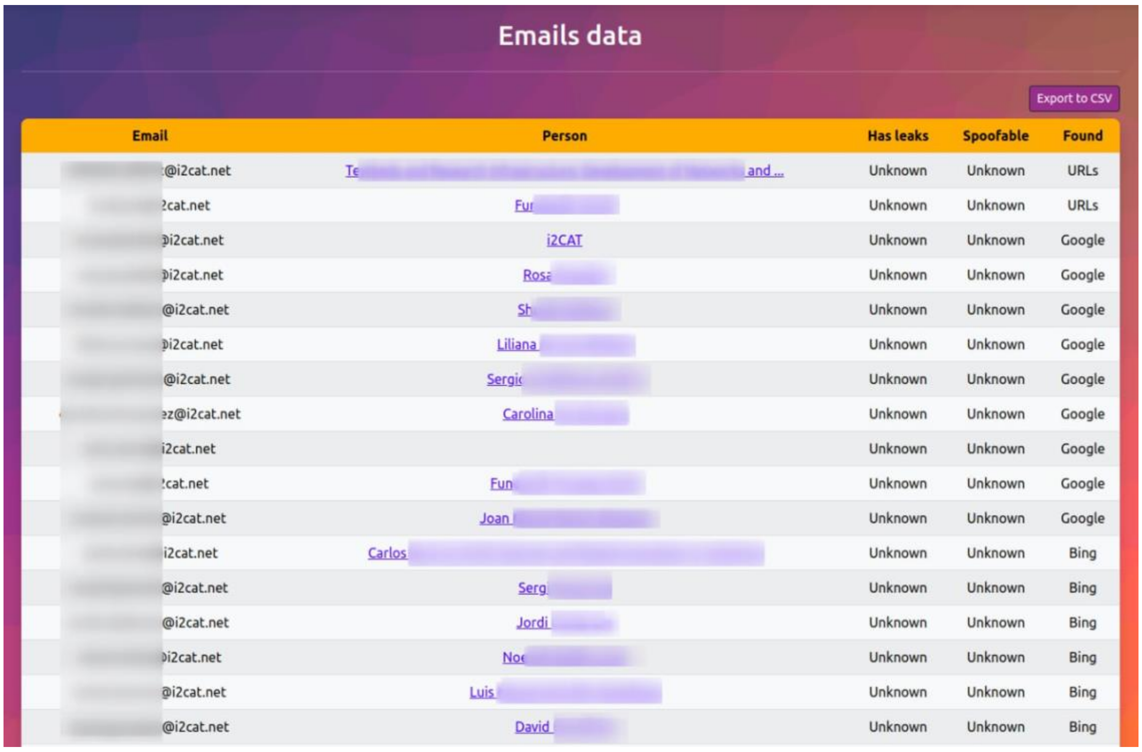
Emails

This tab shows the emails InfoHound has found with some additional information:

- The email
- The Person linked (if found)
- If the email has been found in a leak

- Whether the email is spoofable
- The source where the email was found

Also, a “Export to CSV” button has been added to allow the analyst bring the results to other tools.


Emails data

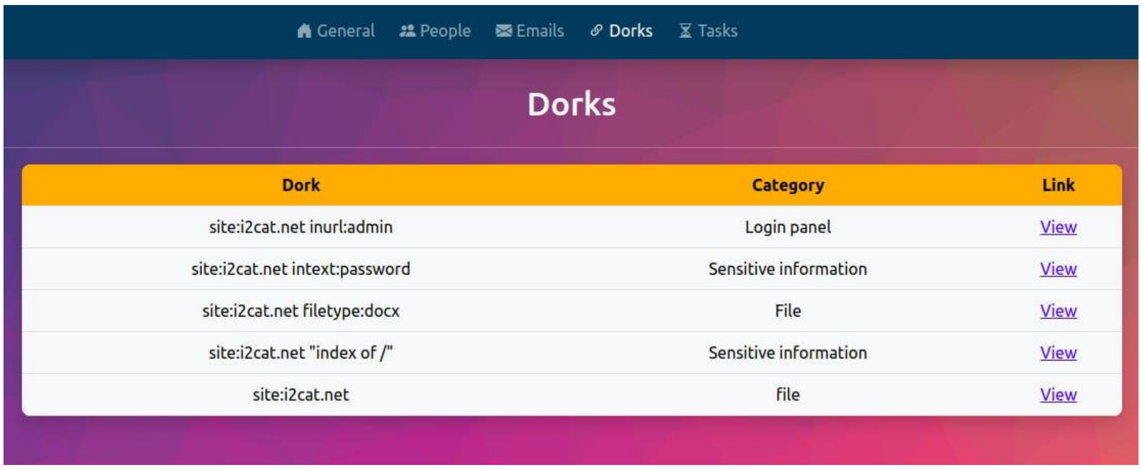
Export to CSV

Email	Person	Has leaks	Spoofable	Found
[redacted]@i2cat.net	Te[redacted] and ...	Unknown	Unknown	URLs
[redacted]cat.net	Fu[redacted]	Unknown	Unknown	URLs
[redacted]i2cat.net	i2CAT	Unknown	Unknown	Google
[redacted]i2cat.net	Rosa[redacted]	Unknown	Unknown	Google
[redacted]@i2cat.net	Sh[redacted]	Unknown	Unknown	Google
[redacted]i2cat.net	Liliana [redacted]	Unknown	Unknown	Google
[redacted]@i2cat.net	Sergi[redacted]	Unknown	Unknown	Google
[redacted]z@i2cat.net	Carolina [redacted]	Unknown	Unknown	Google
[redacted]i2cat.net		Unknown	Unknown	Google
[redacted]cat.net	Fun[redacted]	Unknown	Unknown	Google
[redacted]@i2cat.net	Joan [redacted]	Unknown	Unknown	Google
[redacted]i2cat.net	Carlos [redacted]	Unknown	Unknown	Bing
[redacted]@i2cat.net	Serg[redacted]	Unknown	Unknown	Bing
[redacted]@i2cat.net	Jordi [redacted]	Unknown	Unknown	Bing
[redacted]i2cat.net	Noe[redacted]	Unknown	Unknown	Bing
[redacted]@i2cat.net	Luis [redacted]	Unknown	Unknown	Bing
[redacted]@i2cat.net	David [redacted]	Unknown	Unknown	Bing

Figure 12 Emails list with useful information

Dorks

After thinking of different approaches when designing this tab, the most useful was to only show the dorks that produced results as the analyst could load hundreds or thousands of dorks. So, for every dork with results, the dork will be displayed, the dork’s category and the link to the page that satisfies the dork’s query.



Dorks

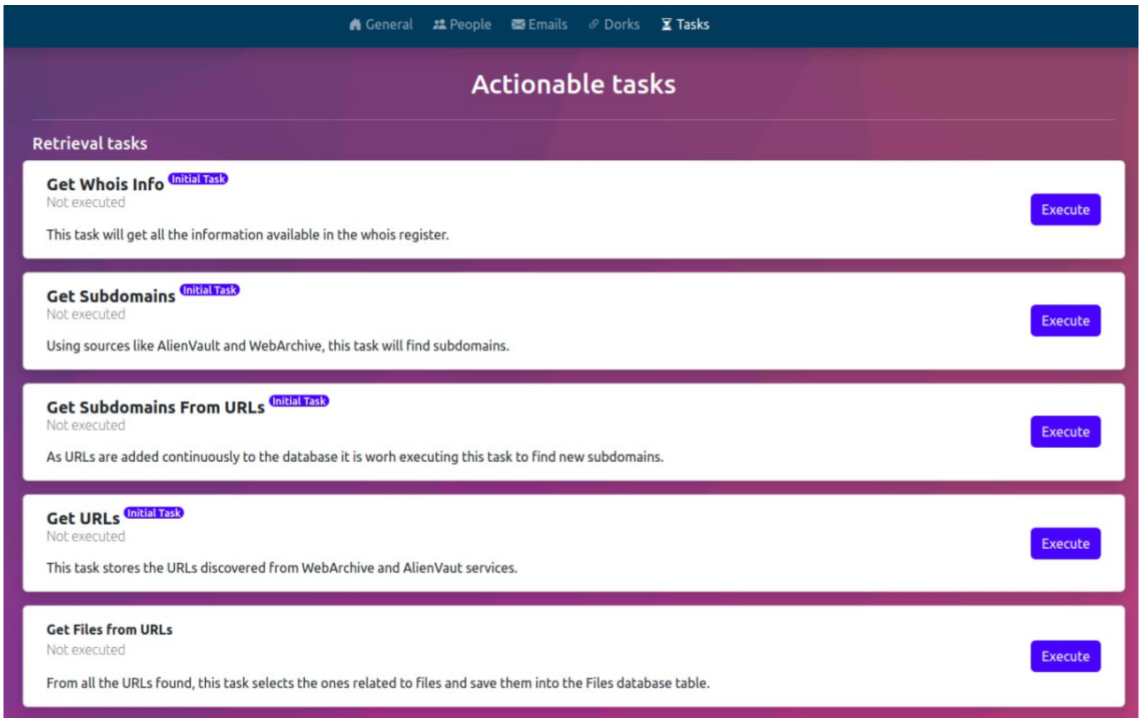
Dork	Category	Link
site:i2cat.net inurl:admin	Login panel	View
site:i2cat.net intext:password	Sensitive information	View
site:i2cat.net filetype:docx	File	View
site:i2cat.net "index of /"	Sensitive information	View
site:i2cat.net	file	View

Figure 13 Dorks table with example values

<u>Tasks</u>

This final tab has the most interactive part of the tool as it handles the launches of the tasks to be performed in the backend. Each task has a name, a description and the time it was last executed. The tasks are divided in retrieval and analysis ones. Also, some badges have been added: initial task and custom.

The initial task indicates that the task will be automatically executed when a domain is added. However, the analyst will be able to re-launch the task in case new something has changed.



Actionable tasks

Retrieval tasks

Get Whois Info
Not executed
This task will get all the information available in the whois register.

Get Subdomains
Not executed
Using sources like AlienVault and WebArchive, this task will find subdomains.

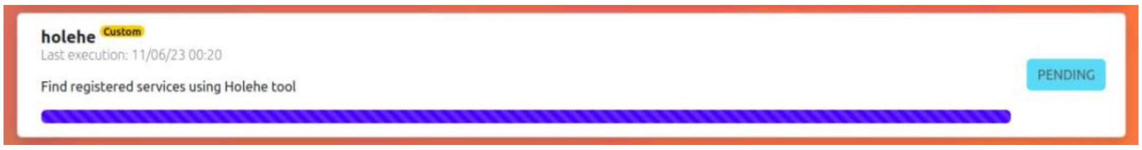
Get Subdomains From URLs
Not executed
As URLs are added continuously to the database it is worh executing this task to find new subdomains.

Get URLs
Not executed
This task stores the URLs discovered from WebArchive and AlienVaut services.

Get Files from URLs
Not executed
From all the URLs found, this task selects the ones related to files and save them into the Files database table.

Figure 14 Tasks and initial tasks

The custom badge indicates that the task has been added by the analyst by the creation of a custom module which will be automatically added as a task.



Find registered services using Holehe tool

Figure 15 Holehe custom task with *Pending* state

It is possible to see the state of a task as the *Execute* button will change to *Pending* and a progress bar will appear when launching a task.

Files and IPs could have their own tab but this idea has been discarded for the following reasons:

- Although IPs could give some potentially valuable information it has been found many domains are running behind services with multiple servers sharing the same IP making it very difficult to extract useful data related to a specific domain.

- We can gather hundreds of Files with InfoHound however, only emails are being extracted at this moment. For this reason, it does not make sense to have a tab specific to Files although it could be added in the future.

5. Results comparison

Before comparing InfoHound with other similar tools we have to stablish which aspects will be evaluated. For instance, the following points have to be considered:

- As InfoHound is a passive focused tool, only these types of reconnaissance techniques will be considered. So, for example, when discovering subdomains, no brute-forcing can be performed.

- Although some tools integrate third party paid services to gather more results only free ones will be used. The main reason is that while they can achieve better results they can be also added to InfoHound as custom modules. The comparison should not be made to something it can be implemented with more time but how the tool gets same results without spending money.

The tools that InfoHound will be tested against are: reconFTW, FinalRecon [34] and theHarvester [35]. These tools are very well known and together they have more than 14.5K stars in GitHub.

For the following tests the domain used is *i2cat.net* as they have let me perform some validations. However, other domains were tested while developing InfoHound, to check that it worked well, and the results are very similar.

5.1. Domain and general information

All three tools and Infohound got the same information as *whois* and *DNS* queries are very standard. However, the way they present it is different. For example, FinalRecon has a specific command argument to display it, reconFTW writes the output in a file and theHarvester shows it directly to through the console.

5.2. Subdomain enumeration

The subdomain enumeration is performed more or less in the same way in all three tools although they get different results:

- FinalRecon searches in CRT.sh [36], ThreatMiner [37], HackerTarget [38], WebArchive and CertSpotter [39]. It obtains 325 unique subdomains.

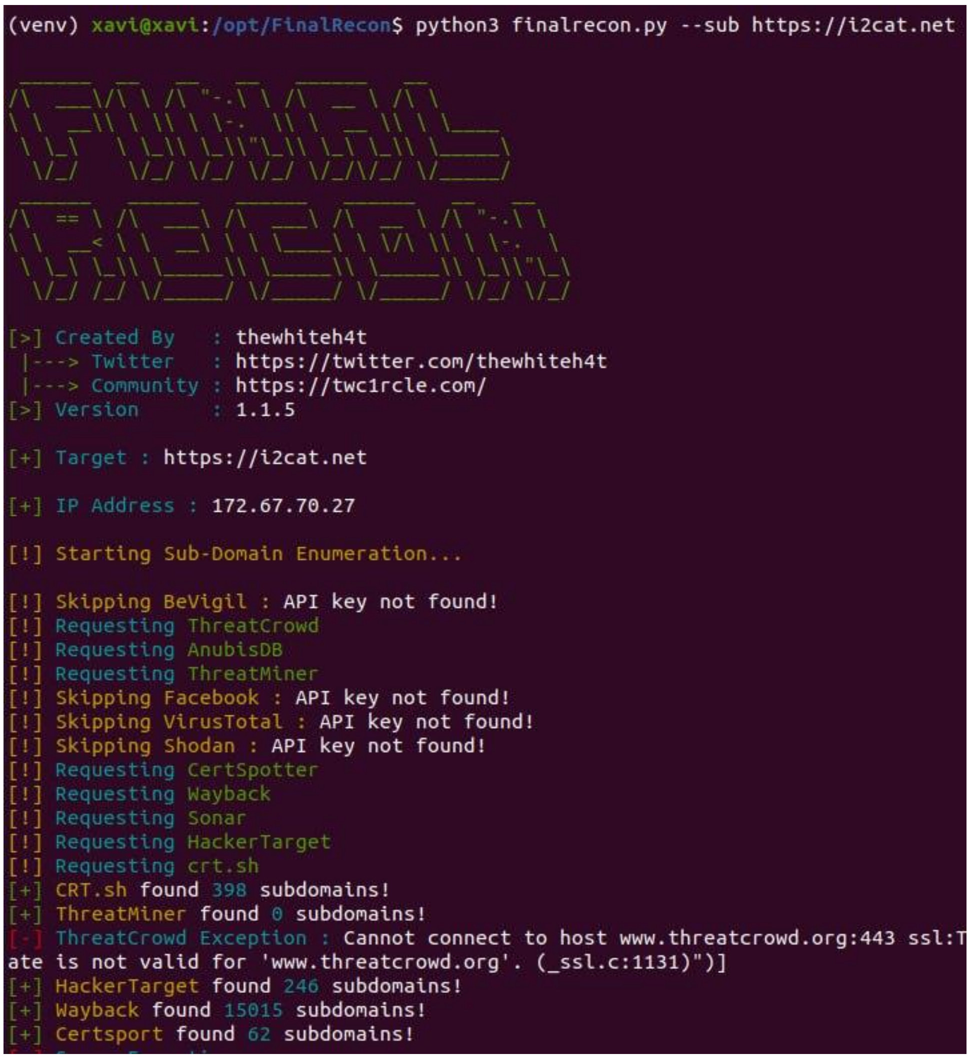
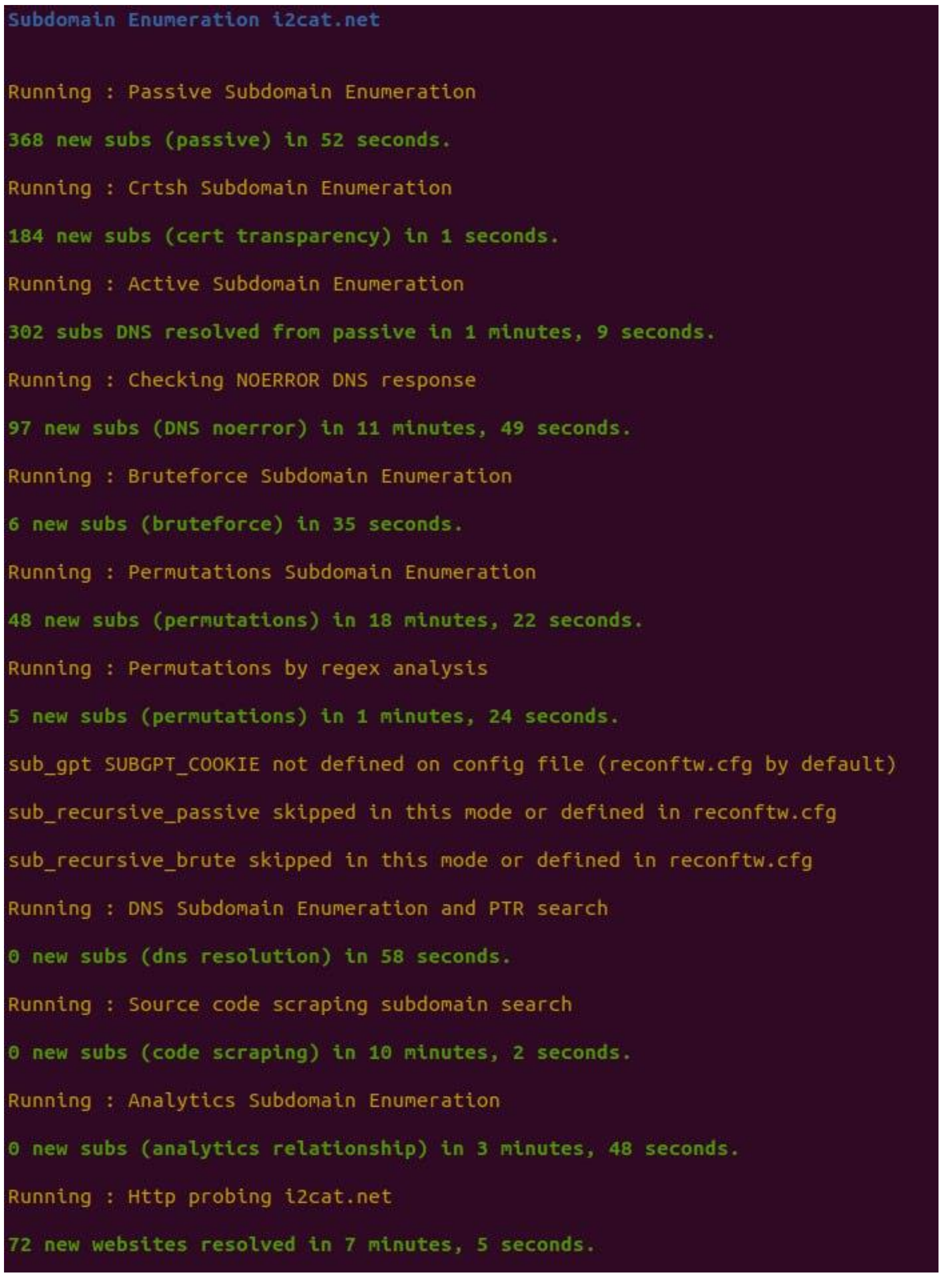

```
(venv) xavi@xavi:/opt/FinalRecon$ python3 finalrecon.py --sub https://i2cat.net

 ______  __   __   __   ______   __
/\  ___\/\ \ /\ "-.\ \ /\  __ \ /\ \
\ \  __\\ \ \\ \ \-.  \\ \  __ \\ \ \____
 \ \_\   \ \_\\ \_\\"\_\\ \_\ \_\\ \_____\
  \/_/    \/_/ \/_/ \/_/ \/_/\/_/ \/_____/

 ______   ______   ______   ______   __   __
/\  == \ /\  ___\ /\  ___\ /\  __ \ /\ "-.\ \
\ \  __< \ \  __\ \ \ \____\ \ \/\ \\ \ \-.  \
 \ \_\ \_\\ \_____\\ \_____\\ \_____\\ \_\\"\_\
  \/_/ /_/ \/_____/ \/_____/ \/_____/ \/_/ \/_/

[>] Created By   : thewhiteh4t
 |---> Twitter   : https://twitter.com/thewhiteh4t
 |---> Community : https://twc1rcle.com/
[>] Version      : 1.1.5

[+] Target : https://i2cat.net

[+] IP Address : 172.67.70.27

[!] Starting Sub-Domain Enumeration...

[!] Skipping BeVigil : API key not found!
[!] Requesting ThreatCrowd
[!] Requesting AnubisDB
[!] Requesting ThreatMiner
[!] Skipping Facebook : API key not found!
[!] Skipping VirusTotal : API key not found!
[!] Skipping Shodan : API key not found!
[!] Requesting CertSpotter
[!] Requesting Wayback
[!] Requesting Sonar
[!] Requesting HackerTarget
[!] Requesting crt.sh
[+] CRT.sh found 398 subdomains!
[+] ThreatMiner found 0 subdomains!
[-] ThreatCrowd Exception : Cannot connect to host www.threatcrowd.org:443 ssl:T
ate is not valid for 'www.threatcrowd.org'. (_ssl.c:1131)")]
[+] HackerTarget found 246 subdomains!
[+] Wayback found 15015 subdomains!
[+] Certsport found 62 subdomains!
```

Figure 16 FinalRecon subdomain enumeration output

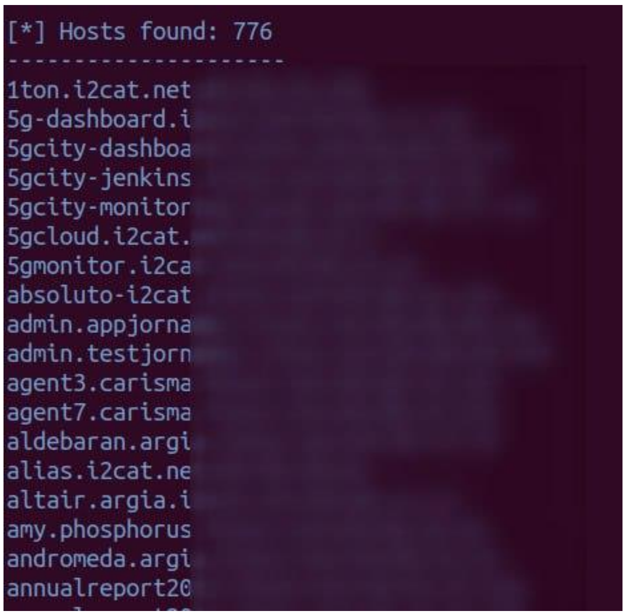
- reconFTW obtained 458 subdomains but <u>368 were passively obtained</u> while the others were got by brute-forcing.



```
Subdomain Enumeration i2cat.net

Running : Passive Subdomain Enumeration
368 new subs (passive) in 52 seconds.
Running : Crtsh Subdomain Enumeration
184 new subs (cert transparency) in 1 seconds.
Running : Active Subdomain Enumeration
302 subs DNS resolved from passive in 1 minutes, 9 seconds.
Running : Checking NOERROR DNS response
97 new subs (DNS noerror) in 11 minutes, 49 seconds.
Running : Bruteforce Subdomain Enumeration
6 new subs (bruteforce) in 35 seconds.
Running : Permutations Subdomain Enumeration
48 new subs (permutations) in 18 minutes, 22 seconds.
Running : Permutations by regex analysis
5 new subs (permutations) in 1 minutes, 24 seconds.
sub_gpt SUBGPT_COOKIE not defined on config file (reconftw.cfg by default)
sub_recursive_passive skipped in this mode or defined in reconftw.cfg
sub_recursive_brute skipped in this mode or defined in reconftw.cfg
Running : DNS Subdomain Enumeration and PTR search
0 new subs (dns resolution) in 58 seconds.
Running : Source code scraping subdomain search
0 new subs (code scraping) in 10 minutes, 2 seconds.
Running : Analytics Subdomain Enumeration
0 new subs (analytics relationship) in 3 minutes, 48 seconds.
Running : Http probing i2cat.net
72 new websites resolved in 7 minutes, 5 seconds.
```

Figure 17 reconFTW subdomain enumeration output

- theHarvester also uses CRT.sh, CertSpotter, ThreatMiner and Alienvault as sources. It launches Sublist3r [40], a subdomain enumeration tool, to obtain as many subdomains as possible. In total, it gets <u>341 unique subdomains</u>.



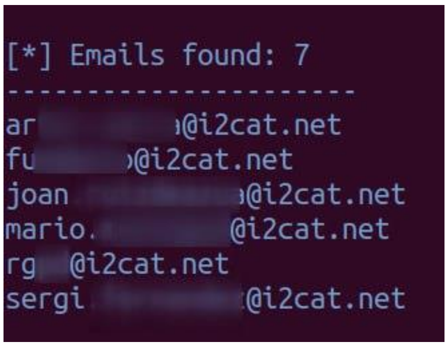
```
[*] Hosts found: 776
--------------------
1ton.i2cat.net [redacted]
5g-dashboard.i [redacted]
5gcity-dashboa [redacted]
5gcity-jenkins [redacted]
5gcity-monitor [redacted]
5gcloud.i2cat. [redacted]
5gmonitor.i2ca [redacted]
absoluto-i2cat [redacted]
admin.appjorna [redacted]
admin.testjorn [redacted]
agent3.carisma [redacted]
agent7.carisma [redacted]
aldebaran.argi [redacted]
alias.i2cat.ne [redacted]
altair.argia.i [redacted]
amy.phosphorus [redacted]
andromeda.argi [redacted]
annualreport20 [redacted]
```

Figure 18 TheHarvester subdomain enumeration output

InfoHound obtained 295 unique subdomains. Although it is lower than other tools it only uses WebArchive, Alienvault, CRT.sh and HackerTarget so there is room to improve by adding other sources. Luckily, this can be done very easily.

5.3. Finding emails

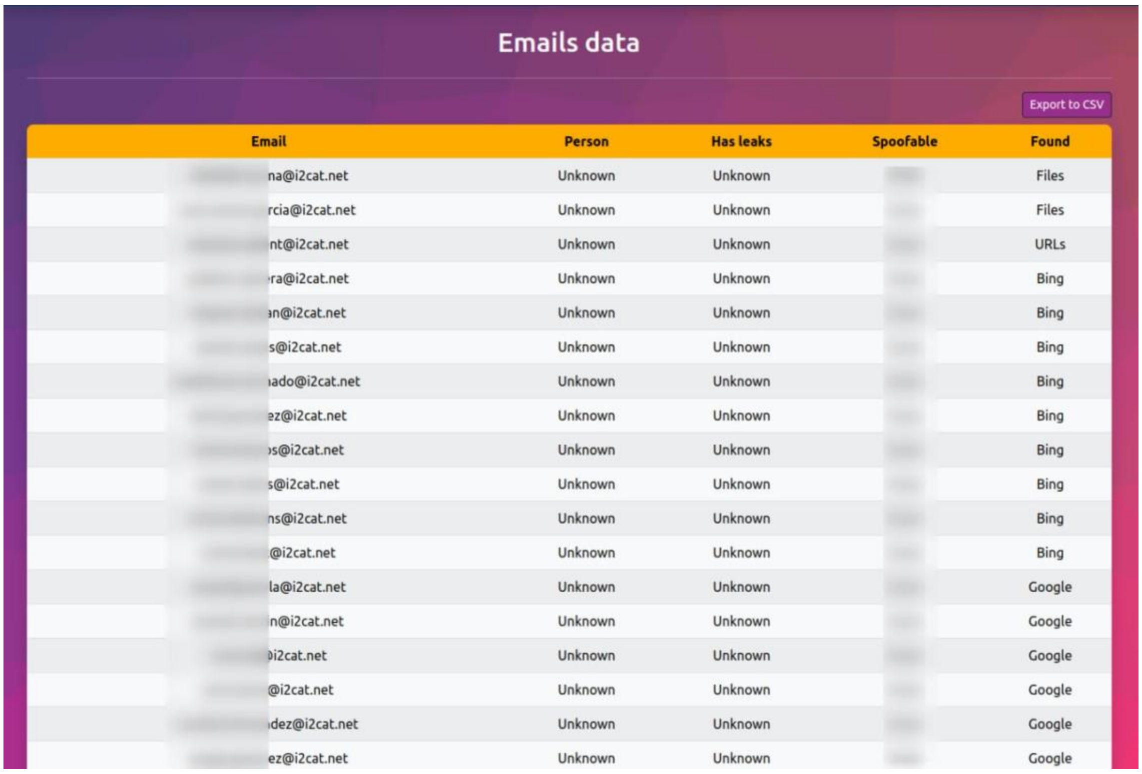
FinalRecon and reconFTW do not have the ability to find emails directly (i.e. they crawl the domain web pages to find relevant data). However, theHarvester does and it obtained a total of 7 emails.



```
[*] Emails found: 7
----------------------
ar[redacted]@i2cat.net
fu[redacted]@i2cat.net
joan[redacted]@i2cat.net
mario.[redacted]@i2cat.net
rg[redacted]@i2cat.net
sergi[redacted]@i2cat.net
```

Figure 19 Emails found by theHarvester

Here, InfoHound surprisingly outperformed by finding 42 emails. This can be due the different locations InfoHound looks for emails. For instance, the main sources are Google (12) and Files (18).



Emails data

Export to CSV

Email	Person	Has leaks	Spoofable	Found
[redacted]na@i2cat.net	Unknown	Unknown	[redacted]	Files
[redacted]rcia@i2cat.net	Unknown	Unknown	[redacted]	Files
[redacted]nt@i2cat.net	Unknown	Unknown	[redacted]	URLs
[redacted]ra@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]n@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]s@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]ado@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]ez@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]s@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]s@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]ns@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]@i2cat.net	Unknown	Unknown	[redacted]	Bing
[redacted]la@i2cat.net	Unknown	Unknown	[redacted]	Google
[redacted]n@i2cat.net	Unknown	Unknown	[redacted]	Google
[redacted]i2cat.net	Unknown	Unknown	[redacted]	Google
[redacted]@i2cat.net	Unknown	Unknown	[redacted]	Google
[redacted]dez@i2cat.net	Unknown	Unknown	[redacted]	Google
[redacted]ez@i2cat.net	Unknown	Unknown	[redacted]	Google

Figure 20 InfoHound - Found emails

5.4. Other entities discovery

This test could not be performed because the tools are unable to correlate results as, for example, they cannot link an email to a person. However, they can perform other straightforward checks like verifying if a subdomain can be taken over or a domain spoofable.

6. Architecture and deployment of the tool

At this stage of the project a lot have been introduced and it may seem difficult to see the structure of the tool and how it works. These are the most important elements of InfoHound:

- Djnago project:
  - Core tool: here lies the code that performs all the gathering and analysis actions. Also, it contains the folders where files are downloaded and custom modules uploaded by the user.
  - WebApp: this element interacts with the database to get the results and display them on a web application so it is easy to understand what has been obtained.
  - PostgreSQL: this is the database used to save all the data found. All actions are performed using Django models so it is abstract for de developer and can the database technology can be changed.
- Task management: Redis and Celery are used to create tasks and get their states. These tasks are created in the WebApp and is the only way to interact with the core tool. However, if a user desires to develop a script to launch the core tool by command line it can be done with minimal effort.

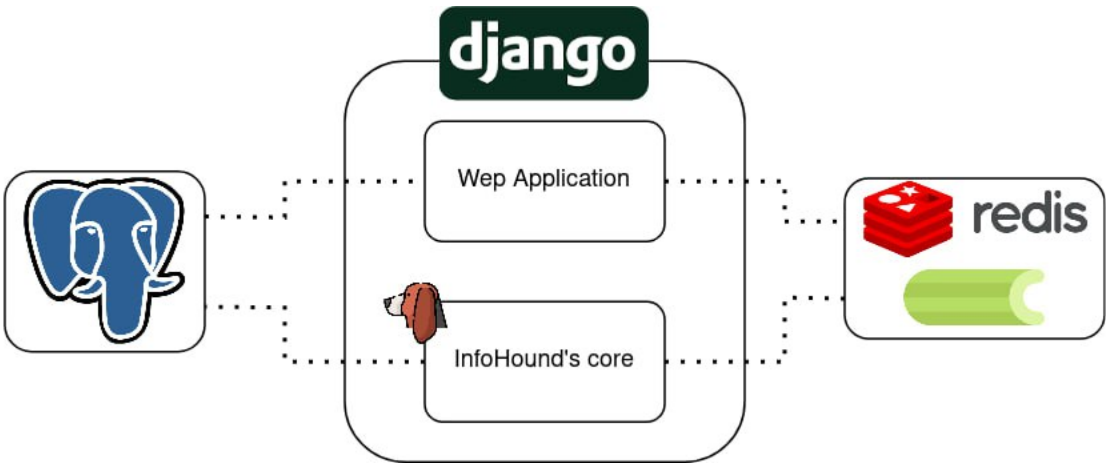
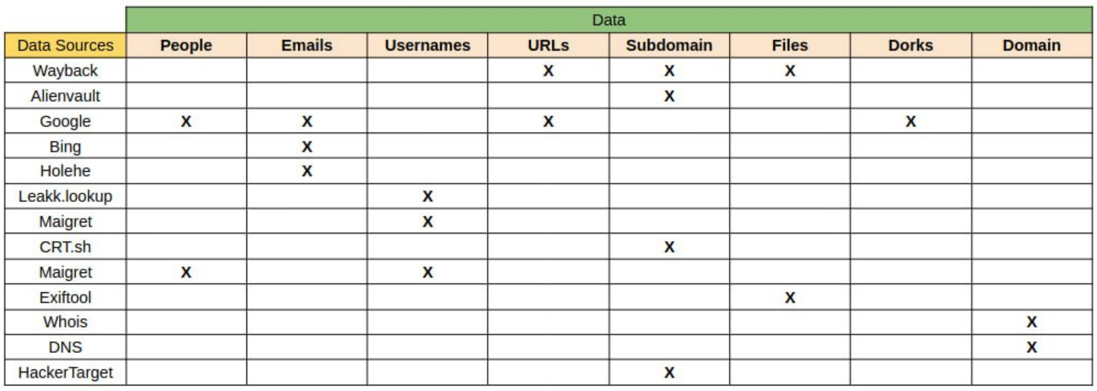


Figure 21 InfoHound arquitecture

In terms of functionalities, the following figure shows the data sources InfoHound has and the data entities they affect:



	Data							
Data Sources	**People**	**Emails**	**Usernames**	**URLs**	**Subdomain**	**Files**	**Dorks**	**Domain**
Wayback				x	x	x		
Alienvault					x			
Google	x	x		x			x	
Bing		x						
Holehe		x						
Leakk.lookup			x					
Maigret			x					
CRT.sh					x			
Maigret	x		x					
Exiftool						x		
Whois								x
DNS								x
HackerTarget					x			

Figure 22 Data entities and sources realation

InfoHound has been thought to be Open Source from the beginning so it is now available in a public GitHub repository [41] licensed under GPL-v3. This license has been chosen for its permissions and conditions.

It has been included a *Dockerfile* and a *docker-compose.yml* files too so it can be deployed easily with Docker. Finally, a GitHub workflow [42] has been created so it automatically builds and publish an image to Dockerhub [43].

7. Conclusions

After analysing the results obtained from testing several domains, it is wise to say that this thesis has become successful in creating a tool that combines several passive enumeration techniques.

It does not only provide results previously gathered by open data sources but it also has its own methods to discover emails, for example. Additionally, InfoHound uses other tools too like Holehe or Maigret to discover more entities related to people. Some uncommon techniques have been implemented like downloading and analysing files, which it is not a feature that the most popular tools have.

However, the main key features are the modularity and customization by adding new modules. The modularity has allowed the InfoHound to its own source for some entities as when URLs are added to the database then they can be used to discover new subdomains. On the other hand, the customization allows the expansion of the tool. There are a lot of open data sources, online paid services or custom created scripts that can bring more rich and updated information. Connecting them to infohound is as simple as creating a Python script and save it to the *custom_modules* folder.

Another worth-mentioning feature is the use of dorks. The dorks are executed by using the Google API to avoid being blocked. Although the free plan has a limitation of 100 queries per day, it is enough to gather interesting information. There are tools, like Pagodo, that can execute dorks without using the Google API but it has to be well configured and the results are obtained very slowly.

7.1. Next steps

There are some things that could be improved, changed or added, here are some of them:

- Improve how people are retrieved. For example, scraping names and other information that could be interesting from social media or services.
- Add more data sources. For example, AIL-Framework [44] (developed by CIRCL [45]) which analyse potential information leaks from unstructured data sources like pastes from Pastebin.
- Add other tools like Blackbird [46], an OSINT tool to search for accounts by username.
- Add a module to retrieve phone numbers.
- Improve the metadata and file analysis so it can show more relevant information like the operating systems and software versions people are using.

- Let the analyst add entries through the web application. Now, if an analyst finds an email and wants to add it to the tool, he/she must do it with PostgreSQL.

- Instead of using regex and other techniques to perform scraping, use Natural Language Processing (NPL) with a library like spaCy [47]. This will improve accuracy and better correlation between entities.

- Connect Maltego with InfoHound. It seems feasible to create a local transformation server for Maltego so given an entity, InfoHound can return related data. The main one would be that given a domain, it will return everything but it could also be applied to emails to find people.

- Improve the web application views. The general tab should be re-designed to be more visually helpful.

- Improve and clean the code so everyone can understand the project internals and contribute to it.

- Use services to determine technologies used in the web domains and adapt the Google Dorks to search for specific misconfigurations and vulnerabilities while creating a web profile.

8. Bibliography

[1] J. Koetsier, “How Google searches 30 trillion web pages, 100 billion times a month”, *https://venturebeat.com/business/how-google-searches-30-trillion-web-pages-100-billion-times-a-month/*, 2013

[2] Wayback Machine: *https://web.archive.org/* (accessed 03/03/23)

[3] A. Martínez, “OSINT - La información es poder”, *https://www.incibe-cert.es/blog/osint-la-informacion-es-poder*, 2014

[4] Google Hacking: What is a Google Hack? *https://www.acunetix.com/websitesecurity/google-hacking/* (accessed 05/03/2023)

[5] MITRE ATT&CK: *https://attack.mitre.org/matrices/enterprise/* (accessed 05/03/2023)

[6] Pagodo: *https://github.com/opsdisk/pagodo* (accessed 06/03/2023)

[7] Hunter: *https://hunter.io/* (accessed 06/03/2023)

[8] Shodan: *https://www.shodan.io/* (accessed 06/03/2023)

[9] FOCA: *https://github.com/ElevenPaths/FOCA* (accessed 06/03/2023)

[10] H8mail: *https://github.com/khast3x/h8mail* (accessed 06/03/2023)

[11] Amass: *https://github.com/OWASP/Amass* (accessed 06/03/2023)

[12] OWASP: *https://owasp.org/* (accessed 07/03/2023)

[13] IntelX: *https://intelx.io/* (accessed 07/03/2023)

[14] ReconFTW: *https://github.com/six2dez/reconftw* (accessed 06/03/2023)

[15] Maltego: *https://www.maltego.com/* (accessed 06/03/2023)

[16] K. Charles, “The Open-Source Intelligence (OSINT) Cycle”, *https://securityboulevard.com/2022/09/the-open-source-intelligence-osint-cycle/*, 2022

[17] R. Scarlett, “Why Python keeps growing, explained” *https://github.blog/2023-03-02-why-python-keeps-growing-explained/*, 2023

[18] Holehe: *https://github.com/megadose/holehe* (accessed 04/04/2023)

[19] Alienvault OTX: *https://otx.alienvault.com/* (accessed 05/04/2023)

[20] Leak-Lookup: *https://leak-lookup.com/* (accessed 05/04/2023)

[21] Have I Been Pwned? *https://haveibeenpwned.com/* (accessed 06/04/2023)

[22] PostgreSQL: *https://www.postgresql.org/* (accessed 06/04/2023)

[23] Django: *https://www.djangoproject.com/* (accessed 06/04/2023)

[24] Whois (Python library): *https://pypi.org/project/python-whois/* (accessed 15/04/2023)

[25] SpoofThatEmail: *https://github.com/v4d1/SpoofThatMail* (accessed 15/04/2023)

[26] EdOverflow, "A Guide To Subdomain Takeovers" *https://www.hackerone.com/application-security/guide-subdomain-takeovers*, 2018

[27] Can I take over XYZ? *https://github.com/EdOverflow/can-i-take-over-xyz* (accessed 15/04/2023)

[28] P. Harvey, ExifTool *https://exiftool.org/*, (accessed 15/04/2023)

[29] Poastal: *https://github.com/jakecreps/poastal*, (accessed 16/04/2023)

[30] Leak-Lookup: *https://leak-lookup.com/*, (accessed 16/04/2023)

[31] Trio: *https://github.com/python-trio/trio*, (accessed 16/04/2023)

[32] Maigret: *https://github.com/soxoj/maigret*, (accessed 16/04/2023)

[33] Sherlock: *https://github.com/sherlock-project/sherlock*, (accessed 16/04/2023)

[34] FinalRecon: https://github.com/thewhiteh4t/FinalRecon, (accessed 29/04/2023)

[35] TheHarvester: https://github.com/laramies/theHarvester, (accessed 10/05/2023)

[36] CRT.sh: https://crt.sh/, (accessed 10/05/2023)

[37] ThreatMiner: https://www.threatminer.org/, (accessed 10/05/2023)

[38] HackerTarget: https://hackertarget.com/, (accessed 10/05/2023)

[39] CertSpotter: https://sslmate.com/certspotter/, (accessed 10/05/2023)

[40] Sublist3r: https://github.com/aboul3la/Sublist3r, (accessed 10/05/2023)

[41] InfoHound – Github: https://github.com/xampla/InfoHound, (accessed 04/06/2023)

[42] Github workflows: https://docs.github.com/en/actions/using-workflows, (accessed 04/06/2023)

[43] InfoHound – Dockehub: https://hub.docker.com/r/xampla/infohound, (accessed 04/06/2023)

[44] AIL-Framework: https://github.com/CIRCL/AIL-framework, (accessed 08/06/2023)

[45] CIRCL: https://www.circl.lu/, (accessed 08/06/2023)

[46] Blackbird: https://github.com/p1ngul1n0/blackbird, (accessed 08/06/2023)

[47] SpaCy: https://spacy.io/, (accessed 09/06/2023)