OSINT Techiques

Telmo Sauce (104428)
DETI
University of Aveiro
Aveiro, Portugal
telmobelasauce@ua.pt

Ricardo Covelo(102668)
DETI
University of Aveiro
Aveiro, Portugal
ricardocovelo11@ua.pt

Renan Ferreira
DETI
University of Aveiro
Aveiro, Portugal
renanaferreira@ua.pt

***Abstract*—Open-source intelligence (OSINT) [1] refers to the process of gathering and analyzing publicly accessible information for multiple objectives, including forensics, security, and investigation. OSINT tools are software programs that assist in the collection, analysis, and presentation of OSINT data. Year after year, these tools are continuously improving in terms of reliability and usefulness, making it imperative for experts in forensics and security to incorporate them into their work [2].**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

OSINT techniques are constantly improving, [3] and integrating into our daily lives. As an illustration of the practical applications of this software, our group has selected three intriguing open-source projects from GitHub. We will delve into each project's use cases, provide setup instructions, offer a demonstration, and elaborate on their underlying functioning. Sherlock and Holehe purposes are the same: from an e-mail or username to search the online activity of a user, obtaining the websites where the credentials were used . They may be used simultaneously since the first one is more general and requires less information (only a user name) but is less precise as multiple may use the same username, while the other is more precise but needs a bit more information.

II. Sherlock

Sherlock [4]is an open-source project that helps users to search for usernames across various social media platforms. Created by a group of developers, it functions as a web-based tool that enables users to input a username and search it against more than 100 platforms to check whether the profile exists or not. It is an excellent solution for internet security researchers, digital professionals, or people who want to check their online presence [5].

A. Purposes

In the context of forensics, Sherlock can be a valuable asset for investigators looking to trace online activities and uncover potential evidence. The software is designed to scout the internet for information related to a specific person or entity, consolidating results from various platforms and websites. This can help investigators piece together a cohesive picture of a person's online activities, relationships, and interests, providing valuable insights and potential leads for a case.

B. Help Files

Sherlock includes two important files to guide users in their search: "sites.md" and "removed_sites.md". The "sites.md" file lists the websites that are currently compatible and actively used by the service. It serves as a reference to identify the platforms that can be searched using Sherlock. On the other hand, the "removed_sites.md" file contains a list of previously supported websites that are no longer compatible for various reasons. These platforms are considered legacy sites that may have changed their structure or policies, making them incompatible with Sherlock's search capabilities. The presence of these files helps users stay updated on the availability and compatibility of different websites when using Sherlock. In addition, users can experiment with their links by adding their personal information to the "data.json" file. This allows them to customize their searches and discover potential matches or connections on websites that they have ownership or control over.

C. Demonstration

The utilization of this tool is very easy to use, for a single user, we just run the command "python3 Sherlock [Username]" where the Sherlock.py is, after downloading the GitHub repository.

Fig. 1. Demonstration of Sherlock with the username covelo13

If the user wants to check various users at the same time he needs to run the command: "python3 Sherlock.py [User1], [User2]..."

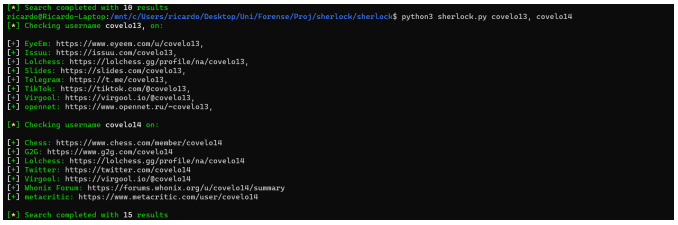


Fig. 2. Demonstration of Sherlock with two names

D. Code analysis

First of all the software checks the number of arguments given to the program by the user as well as the storing format of the output. Then the software does an HTTP request to its own GitHub page to check the last version available and inform the user if the version in the device is not the last one.

```python
# Check for newer version of Sherlock. If it exists, let the user know about it
try:
    r = requests.get(
        "https://raw.githubusercontent.com/sherlock-project/sherlock/master/sherlock/sherlock.py")

    remote_version = str(re.findall('__version__ = "(.*)"', r.text)[0])
    local_version = __version__

    if remote_version != local_version:
        print("Update Available!\n" +
              f"You are running version {local_version}. Version {remote_version} is available at https://github.com/sherlock-project/sherlock")
```

Fig. 3. Example of the HTTP request done by Sherlock to check the last version available of the program

Then the information regarding the Websites available and their URLs is taken from ”resources/data.json” and stored in a dictionary, if the user has chosen to, the NSFW websites are removed from the dictionary.

```python
# Create object with all information about sites we are aware of.
try:
    if args.local:
        sites = SitesInformation(os.path.join(
            os.path.dirname(__file__), "resources/data.json"))
    else:
        sites = SitesInformation(args.json_file)
except Exception as error:
    print(f"ERROR:  {error}")
    sys.exit(1)

if not args.nsfw:
    sites.remove_nsfw_sites()
```

Fig. 4. Information gathering from ”data.json”

After that, the software verifies if the sites provided are supported informing the user if any site of the provided ones is not compatible with the software.

```python
        # User desires to selectively run queries on a sub-set of the site list.
        # Make sure that the sites are supported & build up pruned site database.
        site_data = {}
        site_missing = []
        for site in args.site_list:
            counter = 0
            for existing_site in site_data_all:
                if site.lower() == existing_site.lower():
                    site_data[existing_site] = site_data_all[existing_site]
                    counter += 1
            if counter == 0:
                # Build up list of sites not supported for future error message.
                site_missing.append(f"'{site}'")

        if site_missing:
            print(
                f"Error: Desired sites not found: {', '.join(site_missing)}.")

        if not site_data:
            sys.exit(1)
```

Fig. 5. Verification of the websites provided to ensure compatibility with the system

After all this setup for each of the users, Sherlock runs a function with the same name, this function takes the username that the user wants to check, the sites to verify if the user exists and its information, some flags for specification of use, and a timeout.

```python
for username in all_usernames:

    results = sherlock(username,
                       site_data,
                       query_notify,
                       tor=args.tor,
                       unique_tor=args.unique_tor,
                       proxy=args.proxy,
                       timeout=args.timeout)
```

Fig. 6. Example of usage of the Sherlock function

To ensure a faster search (as it is needed on so many websites) the program relies on threads and parallel computing, taking into account the number of websites to search, the corresponding number of threads are created, to prevent hardware overload, limit the number of threads to a maximum of 20.

```python
# Limit number of workers to 20.
# This is probably vastly overkill.
if len(site_data) >= 20:
    max_workers = 20
else:
    max_workers = len(site_data)
```

Fig. 7. Setting the number of threads

To emulate a legit access to every website a header is added.

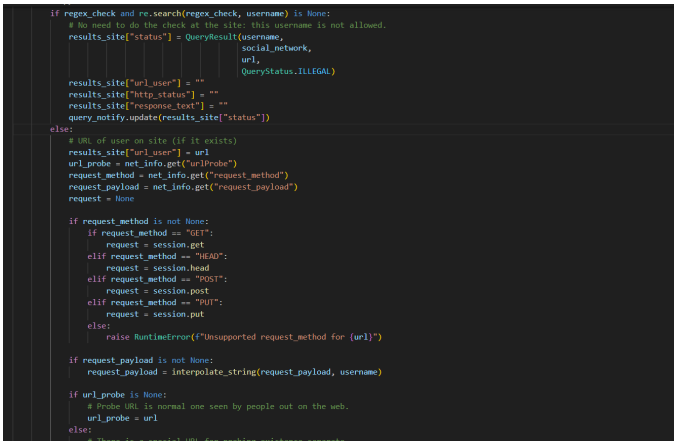
```python
# A user agent is needed because some sites don't return the correct
# information since they think that we are bots (Which we actually are...)
headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:55.0) Gecko/20100101 Firefox/55.0",
}

if "headers" in net_info:
    # Override/append any extra headers required by a given site.
    headers.update(net_info["headers"])

# URL of user on site (if it exists)
url = interpolate_string(net_info["url"], username)
```

Fig. 8. Setting of the headers to be used, in this case, accessing from a computer with a Mac processor and a Mozilla Firefox browser

Then the HTTP request is made, then depending on the response obtained the software can have different types of reaction: If the response is null, (There is no user associated with the username) the information of that website in the previously mentioned dictionary is set to null, if it is not, the URL to the user page on the website is stored in the dictionary.



```
if regex_check and re.search(regex_check, username) is None:
    # No need to do the check at the site: this username is not allowed.
    results_site["status"] = QueryResult(username,
                                         social_network,
                                         url,
                                         QueryStatus.ILLEGAL)
    results_site["url_user"] = ""
    results_site["http_status"] = ""
    results_site["response_text"] = ""
    query_notify.update(results_site["status"])
else:
    # URL of user on site (if it exists)
    results_site["url_user"] = url
    url_probe = net_info.get("urlProbe")
    request_method = net_info.get("request_method")
    request_payload = net_info.get("request_payload")
    request = None

    if request_method is not None:
        if request_method == "GET":
            request = session.get
        elif request_method == "HEAD":
            request = session.head
        elif request_method == "POST":
            request = session.post
        elif request_method == "PUT":
            request = session.put
        else:
            raise RuntimeError(f"Unsupported request_method for {url}")

    if request_payload is not None:
        request_payload = interpolate_string(request_payload, username)

    if url_probe is None:
        # Probe URL is normal one seen by people out on the web.
        url_probe = url
    else:
```

Fig. 9. Checking if the request is valid and posterior action

E. limitations

Although this software has an enormous list of pros, such as its efficacy, lightweight, speed, and ease of use, it also has some serious cons that cannot be ignored, and although some might have some workarounds it weights in the overall user experience as well as its efficacy.

- **False positives** As may be easy to foresee this setback, it may be impossible to circumvent. The fact that other users may use the same username that our subject uses on other websites, may misdirect the one using the software. The user needs to check if the user of each one of the websites is indeed the same person as everyone.
- **Clear web** Due to the fact that the Dark net URLs are changing constantly, we can't use Sherlock on this kinda of website. [6]

III. Holehe

Holehe [7], similar to Sherlock, is an open-source project designed to assist users in identifying platforms associated with their email addresses. Developed by "megadose", a reputable developer of numerous open-source intelligence (Osint) tools, Holehe serves as a valuable resource for professionals in the field. By inputting a target email address, users can determine the online presence and digital footprints associated with a specific individual or even their email. [8]

A. Purposes

Holehe is a powerful Osint tool that serves multiple purposes in conducting investigations. With just an email address, this tool not only provides insights into an individual's online presence but also facilitates further investigations by revealing additional websites to explore.

Additionally, Holehe seamlessly integrates with tools like Sherlock II, enabling users to discover more usernames and expand their search parameters. In summary, Holehe simplifies Osint inspections, enhances the investigation process, and amplifies the scope of the search.

B. Demonstration

To make use of the Holehe tool, the first step is to install it by running the command "pip3 install Holehe". Once installed, the tool can be executed by running the command "Holehe test@mail.com". This command will initiate the search for associated websites and provide a comprehensive response.

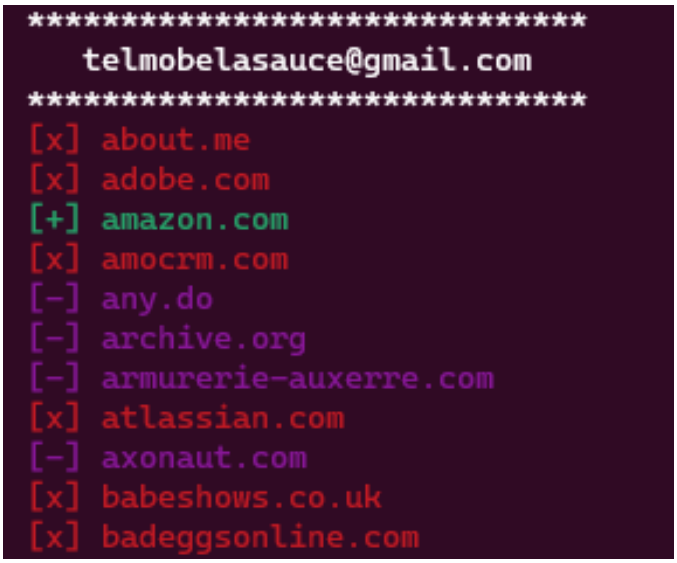


Fig. 10. Enter Caption

Additionally, Holehe offers the functionality to filter the output and display only the platforms with a positive response. To view the available flags, the command "holehe -h" can be used. To replicate the response shown in Figure **??**, run the command "holehe telmobelasauce@gmail.com –only-used".

From the response we can say that the user "telmobelasauce@gmail.com" got an account on websites like spotify, firefox and others.

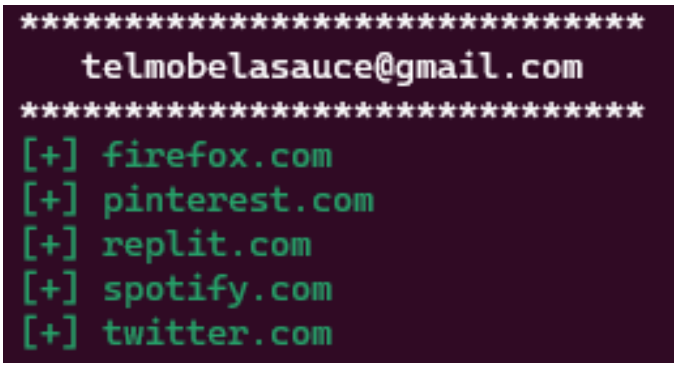


Fig. 11. Enter Caption

C. Tool Functionality

Holehe utilizes three key functionalities of websites: "login," "register," and "Forgot Password." By sending HTTP requests based on these functionalities and analyzing the responses, Holehe determines whether an account exists in the database or not.

It's important to note that Holehe does not actually reset any passwords or creates accounts. It solely relies on the responses from the platforms to gather information, making it a non-intrusive tool for open-source intelligence.

To illustrate this process, consider the following examples:

```
response = await client.post('https://about.me/n/signup', headers=headers, data=data)
if response.status_code == 409:
    out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                "rateLimit": False,
                "exists": True,
                "emailrecovery": None,
                "phoneNumber": None,
                "others": None})
elif response.status_code == 200:
    out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                "rateLimit": False,
                "exists": False,
                "emailrecovery": None,
                "phoneNumber": None,
                "others": None})
else:
    out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                "rateLimit": True,
                "exists": False,
                "emailrecovery": None,
                "phoneNumber": None,
                "others": None})
```

Fig. 12. About.me Register Request

```
try:
    url = "https://www.amazon.com/ap/signin?openid.pape.max_auth_age=0&openid.return_to=https%3A%2F%2Fwww.am
    req = await client.get(url, headers=headers)
    body = BeautifulSoup(req.text, 'html.parser')
    data = dict([(x["name"], x["value"]) for x in body.select(
        'form input') if ('name' in x.attrs and 'value' in x.attrs)])
    data["email"] = email
    req = await client.post(f'https://www.amazon.com/ap/signin/', data=data)
    body = BeautifulSoup(req.text, 'html.parser')
    if body.find("div", {"id": "auth-password-missing-alert"}):
        out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                    "rateLimit": False,
                    "exists": True,
                    "emailrecovery": None,
                    "phoneNumber": None,
                    "others": None})
    else:
        out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                    "rateLimit": False,
                    "exists": False,
                    "emailrecovery": None,
                    "phoneNumber": None,
                    "others": None})
```

Fig. 13. Amazon Login Request

```
headers['X-IMS-Authentication-State-Encrypted'] = r.headers['x-ims-authentication-state-encrypted']
params = {
    'purpose': 'passwordRecovery',
}
response = await client.get(
    'https://auth.services.adobe.com/signin/v2/challenges',
    headers=headers,
    params=params)
response=response.json()
if 'errorCode' in response:
    out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                "rateLimit": False,
                "exists": True,
                "emailrecovery": None,
                "phoneNumber": None,
                "others": None})
else:
    out.append({"name": name,"domain":domain,"method":method,"frequent_rate_limit":frequent_rate_limit,
                "rateLimit": False,
                "exists": True,
                "emailrecovery": response['secondaryEmail'],
                "phoneNumber": response['securityPhoneNumber'],
                "others": None})
```

Fig. 14. Adobe.com PasswordRevory Request

Holehe is designed to be efficient and user-friendly. It runs asynchronously, which means it can send out multiple requests at the same time and does not need to wait for a response before moving on to the next platform. This allows Holehe to check over 120 platforms in a short amount of time.

To make this possible it uses Trio [9] which is a Python library that allows multiple tasks to run at the same time, improving efficiency and HTTPX [10] which is another Python library that handles sending HTTP requests. When used together, they can send multiple HTTP requests at the same time.

The typical flow of the Holehe tool begins with email validation, followed by loading the modules for each platform that Holehe can check. Each module contains the necessary code to determine if the target email is associated with a specific platform. AfterWards, HTTPX is initialized to send asynchronous HTTP requests, and Trio is used to run the modules concurrently. Finally, the results are sorted and HTTPX is closed, allowing the output to be printed. If the "–csv" flag is used, the tool generates a file to export the output.

```
if not is_email(email):
    exit("[-] Please enter a target email ! \nExample : holehe email@example.com")

# Import Modules
modules = import_submodules("holehe.modules")
websites = get_functions(modules,args)
# Get timeout
timeout=args.timeout
# Start time
start_time = time.time()
# Def the async client
client = httpx.AsyncClient(timeout=timeout)
# Launching the modules
out = []
instrument = TrioProgress(len(websites))
trio.lowlevel.add_instrument(instrument)
async with trio.open_nursery() as nursery:
    for website in websites:
        nursery.start_soon(launch_module, website, email, client, out)
trio.lowlevel.remove_instrument(instrument)
# Sort by modules names
out = sorted(out, key=lambda i: i['name'])
# Close the client
await client.aclose()
# Printh te result
print_result(out,args,email,start_time,websites)
credit()
# Export results
export_csv(out,args,email)
```

Fig. 15. Core code flow

IV. H8MAIL

h8mail is an email OSINT and breach hunting tool [11] using different breach and reconnaissance services, or local breaches such as Troy Hunt's "Collection1" and the infamous "Breach Compilation" torrent. [12] [13]

Its main features are:

- Easy installation.
- Email pattern matching.
- Pass multiple files or folders containing emails or URLs as parameters.
- Loosey patterns for local searches ("john.smith", "evil-corp").
- Bulk file-reading for targeting.
- Output to CSV file or JSON
- Compatible with the "Breach Compilation" torrent scripts [14].
- Search cleartext and compressed .gz files locally using multiprocessing.
- Compatible with "Collection#1".
- Get related emails.

- Chase related emails by adding them to the ongoing search.
- Supports premium lookup services for advanced users
- Custom query premium APIs. Supports username, hash, ip, domain and password and more
- Regroup breach results for all targets and methods
- Includes option to hide passwords for demonstrations

A. Main services

One of the best benefits of *h8mail* is the integration with several other *OSINT* tools, even though it is important to notice that not all of them are freely available or functioning.

- ***HaveIBeenPwned(v3)***: It presents the number of email breaches. It requires an API key.
- ***HaveIBeenPwned Pastes(v3)***: It presents URLs of text files mentioning targets. It requires an API Key.
- ***Hunter.io(Public)***: It presents the number of related emails.
- ***Hunter.io(Private free tier)***: It presents clear-text related emails and chasing. It requires an API key.
- ***Snusbase***: It presents clear-text passwords, hashes and salts, usernames, and IP addresses. It is a high-speed API and requires an API key.
- ***Leak-Lookup(Public)***: It provides the number of searchable breach results. It requires an API key.
- ***Leak-Lookup(Private)***: It provides clear-text passwords, hashes and salts, usernames, IP addresses, and domains. It requires an API key.
- ***WeLeakInfo(Public)***: Currently offline.
- ***WeLeakInfo(Private)***: Currently offline.
- ***Emailrep.io***: It provides last seen in breaches and social media profiles. It requires an API key.
- ***Scylla.io***: It provides clear-text passwords, hashes and salts, usernames, IP addresses, and domains. It requires an API key.
- ***Dehashed.com***: It provides clear-text passwords, hashes and salts, usernames, IP addresses, and domains. It requires an API key.
- ***IntelX.io***: It provides clear-text passwords, hashes and salts, usernames, IP addresses, domains, Bitcoin wallets, and IBAN numbers. It requires an API key.
- ***Breachdirectory.org***: It provides clear-text passwords, hashes and salts, usernames, IP addresses, and domains. It requires an API key.

B. Command Line Interface

H8mail can be deployed as a Python package, which can be executed in the command line with the following arguments:

- **-h/–help**: Shows help message and exits.
- **-t/–targets**: Targets, which are either string inputs or files.
- **-u/–url**: URLs, which are either string inputs or files. Requires 'http://' or 'https://' in the URL.
- **-q/–custom-query**: Performs a custom query. Supports username, password, IP address, hash, domain. Performs an implicit "loose" search when searching locally [15].
- **–loose**: Allows loose search by disabling email pattern recognition. Uses spaces as pattern separators.
- **-c/–config**: Configuration file for API keys. Accepts keys from *Snusbase*, *WeLeakInfo*, *Leak-Lookup*, *HaveIBeenPwned*, *Emailrep*, *Dehashed* and *Hunter.io*.
- **-o/–output**: File to write *CSV* output.
- **-j/–json**: File to write *JSON* output.
- **-bc/–breachcomp**: Path to the *breachcompilation* torrent folder.
- **-sk/–skip-defaults**: Skips *Scylla* and *Hunter.io* check. Ideal for local scans.
- **-k/–apikey**: Passes configuration options. Supported format: "K=V, K=V".
- **-lb/–local-breach**: Local clear-text breaches to scan for targets. Uses multiprocessing, one separate process per file, on a separate worker pool by arguments [16].
- **-gz/–gzip**: Local *tar.gz*(*gzip*) compressed breaches to scans for targets. Works exactly like a local search. It looks for '*gz*' in the filename.
- **-sf/–single-file**: If the breach contains big clear-text or *tar.gz* files, set this flag to view the progress bar. Disables concurrent file searching for stability.
- **-ch/–chase**: Add related emails from *Hunter.io* to ongoing target list. It defines the number of emails per target to chase. Requires *Hunter.io* private API key if used without power-chase flag [17].
- **–power-chase**: Add related emails from ALL API services to the ongoing target list. Uses with chase flag.
- **–hide**: Only shows the first 4 characters of found passwords to output.
- **–debug**: It prints request debug information.
- **-g/–gen-config**: Generates a configuration file template in the current working directory and exits.

C. Code analysis

The *h8mail* program can be considered a Python package. Its main functionalities' code is in the sub-folder *h8mail*, which contains the sub-folder *utils*, with the main implementation.

The main modules that compose the ***utils*** folder are:

- ***gen_config***: It contains the function ***gen_config_file()***, which generates a template file to store the necessary information, like email, password, and API key, that can be used as the configuration file for later purposes, in the file path *h8mail_config.ini*.
- ***url***: Its purpose is to support file parsing to obtain URLs and to parse URL pages to fetch emails.
- ***helpers***: It contains functions that support the program functioning.
- ***classes***: It contains the definition and implementation of two main classes that are used throughout the code: ***local_breach_target***, containing a local search breach data, and ***target***, which encapsulates target information and the methods to call third-party APIs, like ***get_hibp3(self, api_key)***, and ***get_hunterio_public(self)*** [18].

- ***localsearch*** & ***localgzipsearch***: Modules for local searches processing in clear-text, and *gzip* compressed files, respectively. It executes both in multiprocessing and single-process environments.
- ***chase***: It contains the function ***chase(target, user_args)***, which creates a list of new targets related to *target*, which allows recursive target search.
- ***intelx***: Module to manage the interaction with the *IntelX* API.
- ***breachcompilation***: It manages the breach compilation processing.
- ***run***: Contains the necessary functions to process the command line arguments, manage the user interaction and compose the search process. Its main function is ***target_factory(targets, user_args)***, which receives a list of targets and executes a search through each one of them.

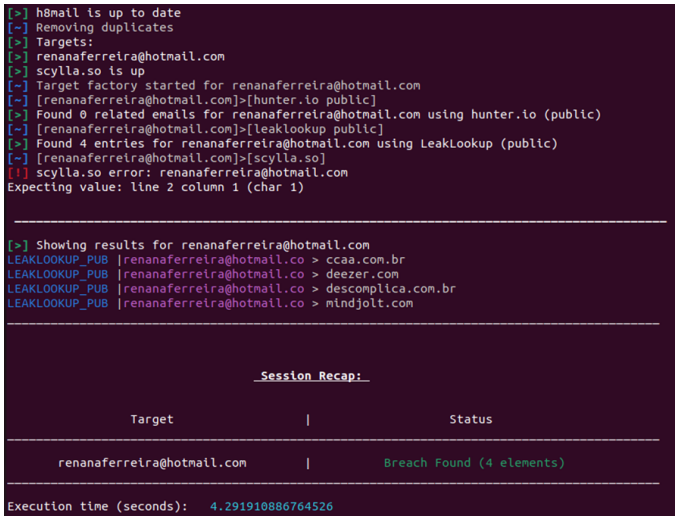


Fig. 16. Demonstration 3 results

V. Demonstration

We conducted a series of command execution with different parameters in order to test the functioning of ***h8mail***.

1) **Definition**: Search with multiple files containing targets
Command:
```
$ h8mail -t all-targets.txt
↪ all-targets-2.txt
```
Observation: It gives an error by the *scylla* API.

2) **Definition**: Generate configuration template file
Command:
```
$ h8mail --gen-config -t
↪ dir-targets/all-targets.txt
```
3) **Definition**: Search with configuration file (For *LeakLookup* Public API) and saving results in *CSV* file.
Command:
```
$ h8mail -t
↪ renanaferreira@hotmail.com -c
↪ h8mail_config.ini -o
↪ results1.csv
```

4) **Definition**: Search in url.
Command:
```
$ h8mail -u
↪ https://ebec.bestporto.org/
```
Observation: The program considers source files as email addresses, and does not recognize emails in *a* tags.

5) **Definition**: Local Search at multiprocessing environment.
Command:
```
$ h8mail -t renanaferreira@ua.pt
↪ telmosauce@gmail.com -lb
↪ local-src.txt
```

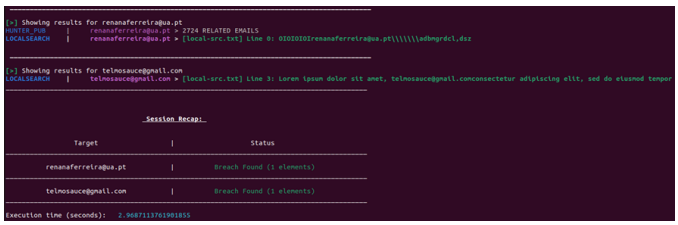


Fig. 17. Demonstration 5 results

6) **Definition**: Local Search at single-process environment.
Command:
```
$ h8mail -t renanaferreira@ua.pt
↪ telmosauce@gmail.com -lb
↪ local-src.txt {single-file
```
7) **Definition**: *Gzip* local search at single-process environment.
Command:
```
$ h8mail -t renanaferreira@ua.pt
↪ telmosauce@gmail.com -gz
↪ local-src.txt.gz
```

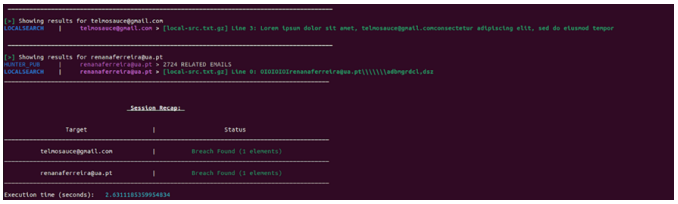


Fig. 18. Demonstration 7 results

8) **Definition**: Search by skipping defaults and saving the results in a *json* file.
Command:

```
$ h8mail -t
↪  renanaferreira@hotmail.com -c
↪  h8mail_config.ini --json
↪  results2.json --skip-defaults
```

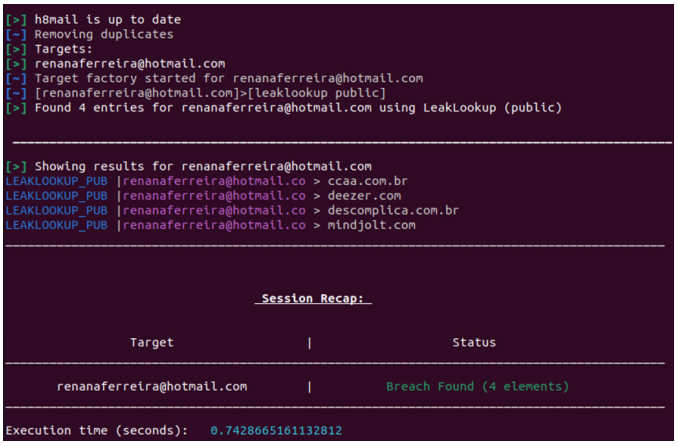


Fig. 19. Demonstration 8 results

9) **Definition**: Local search with debugging option.
Command:

```
$ h8mail -t renanaferreira@ua.pt
↪  telmosauce@gmail.com -lb
↪  dir-local-search {debug
```

Observation: It gives an error by debugging a local search.

Fig. 20. Demonstration 9 results

10) **Definition**: Search with chase limit of 2 and power-chase option.
Command:

```
$ h8mail -t
↪  renanaferreira@hotmail.com -c
↪  h8mail_config.ini -o
↪  results4.csv --chase 2
↪  --power-chase
```

Observation: It recognizes non-email data as email, which is a clear error.

11) **Definition**: Local search by loose target identifier.
Command:

```
$ h8mail -t renanaferreira
↪  telmosauce -lb
↪  dir-local-search
↪  --skip-defaults {loose
```

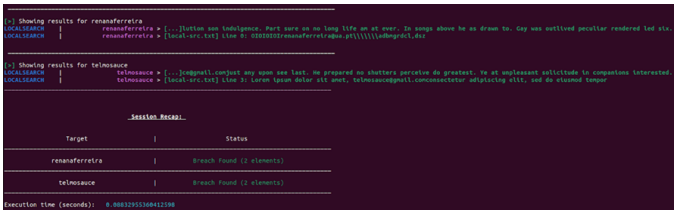


Fig. 21. Demonstration 11 results

Those experiments were able to present that h8mail works relatively well, offering an efficient *wrapping* of different *OSINT* tools, and also managing the input and arguments, but it also presents strong errors, like legacy API URLs, i.e. *Scylla*, and exceptions caused by bad designed code, like the debugging in the local search, but for an open-source free tool, it is extremely well designed and proper implemented, with a clean and documented code.

VI. Conclusion

To sum up, these tools together form a comprehensive package for OSINT applications. Sherlock and Holehe, with their emphasis on social media and email platforms respectively, provide a wide range of abilities for gathering data. Additionally, h8mail complements these tools by facilitating breach investigation and thorough reconnaissance. Although they do have certain limitations, their effectiveness, user-friendly interfaces, and ongoing enhancements establish them as indispensable assets in the realms of digital forensics and cybersecurity investigations.

REFERENCES

[1] M. Bazzell, *Open Source Intelligence Techniques*, February 24, 2012.

[2] E. Casey, *Digital Forensics and Cybersecurity*, October 7, 2009.

[3] J. Pastor-Galindo, "The not yet exploited goldmine of osint: Opportunities, open challenges and future trends," 09 January 2020.

[4] "sherlck." [Online]. Available: https://github.com/sherlock-project/sherlock

[5] S. A. khateeb Nitin Agarwal, "Social cyber forensics: leveraging open source information and social network analysis to advance cyber security informatics," 19 July 2019.

[6] B. Percy, *Dark Net*, 30 October 2017.

[7] megadose, "Holehe," 2021. [Online]. Available: https://github.com/megadose/holehe

[8] D. McCaddon, *Following Digital Footprints*, 29 July, 2016.

[9] Python-trio, "Trio: Python library for async concurrency and i/o." [Online]. Available: https://github.com/python-trio/trio

[10] Encode, "Httpx: Http client for python." [Online]. Available: https://github.com/encode/httpx

[11] R. D. Steele, "Open source intelligence," *Handbook of intelligence studies*, vol. 42, no. 5, pp. 129–147, 2007.

[12] S. Davidoff, "Data breaches: Crisis and opportunity," October 2019.

[13] M. Glassman and M. J. Kang, "Intelligence in the internet age: The emergence and evolution of open source intelligence (osint)," *Computers in Human Behavior*, vol. 28, no. 2, pp. 673–682, 2012.

[14] N. A. Hassan and R. Hijazi, *Open source intelligence methods and tools*. Springer, 2018.

[15] D. R. Hayes and F. Cappa, "Open-source intelligence for risk assessment," *Business Horizons*, vol. 61, no. 5, pp. 689–697, 2018.

[16] B. Akhgar, P. S. Bayerl, and F. Sampson, *Open source intelligence investigation: from strategy to implementation*. Springer, 2017.

[17] D. Trottier, "Open source intelligence, social media and law enforcement: Visions, constraints and critiques," *European Journal of Cultural Studies*, vol. 18, no. 4-5, pp. 530–547, 2015.

[18] B.-J. Koops, J.-H. Hoepman, and R. Leenes, "Open-source intelligence and privacy by design," *Computer Law & Security Review*, vol. 29, no. 6, pp. 676–688, 2013.