SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

Engineer thesis

Design and implementation of natural language sensitive data extraction tool

author: Maksym Brzęczek

supervisor: Błażej Adamczyk, PhD

consultant: Michał Kawulok, DSc PhD

Gliwice, January 2020

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 18 stycznia 2020

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Design and implementation of natural language sensitive data extraction tool" spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 18 stycznia 2020

.....

(podpis promotora)

Contents

Chapter 1

Introduction

This chapter presents the problem that the project tries to solve and describes the scope of the thesis. It also describes the document structure.

1.1 Description of the problem

The invention and propagation of the Internet has boosted the ways in which technology impacts all of us. In this age an increasing amount of our everyday life is digitized and dependent on cybernetic systems hosted and operated by independent corporations and institutions. Significant amount of our tasks has become automated through information technology solutions. The phisical world surrounding us also becomes intertwined with technology. We are gradualy connecting things of everyday use like cars or house locks to the web through Internet of Things technologies. This process has made our lives simpler and allowed us to achieve amazing things but it has also made us vulnerable to cybernetic attacks. It is only natural that the rise of the impact of technology was followed by the rise in the cyber crime and cyber security providers [16].

Over the years an ecosystem has emerged that constantly competes with malicious hackers to keep us all secure. One of the elements of this structure is penetration testing also known as ethical hacking. In it's core, this practice is simply simulating a real attack. There are multiple sources that depict approaches used to perform this process. One of the common denominators between all of them is the

importance of gathering information [14]. The reason for that is because the more information an attacker can uncover and analyze, the bigger the chance of finding vulnerable systems or flaws in them. One of the clusters of information in companies is a communication channel like slack or discord. There are many situations where employees share information connected to projects and their workplace enviroment. If an attacker was to access such a platform he could potentially analyse the conversation history in search of sensitive information like ip addresses, logins, passwords, emails, phone numbers, etc.

Such information is especially important from a legal point of view. Introduction of General Data Protection Regulation in 2016 has put a preassure on many legal bodies to responsibly handle people's personal data under a threat of heavy financial penalties [22]. Monitoring of the data located in the private entity's internal communication channel might prove very useful in fulfilling the legislative requirements of private information processing.

Unfortunately such a task may be very time and resource consuming. A tool capable of scanning the history of communication channel in search of data that would meet some established criteria could however fulfill this job or at least increase efficiency of the person responsible for it.

1.2 Project scope

This thesis focuses on the research into proper implementation of a cyber security tool with the purpose of processing a large amount of natural language messages in search of data that can be classified as personal or sensitive from the cyber security point of view. Special focus is put on possibility of calibration and customisation of the search engine and it's reusability, regardless of circumstances and enviroment.

The project will primarily focus on finding nine categories of information:

- IP v4 addresses - numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication [12].
- IP v6 addresses - 32 to 128 bits identifiers for interfaces and sets of interfaces implementing Internet Protocol version 6 [19][18].

- National identification numbers - numbers issued by a Government Entity as a means of providing Identification of their citizens and/or foreigners [23]. In this thesis it is frequently refered to as social security number.
- ID card numbers - serial numbers of documents (such as a cards) bearing identifying information about and often a photograph of the individual whose name appears on it [3].
- MAC addresses - unique identifier for an Ethernet or network adapter over a network. It distinguishes different network interfaces and is used for a number of network technologies, particularly most IEEE 802 networks, including Ethernet [9].
- Domain names - combinations of letters and numbers used in combination of the various domain name extensions, such as .com, .net to find and identify computers on the Internet [11].
- Email addresses - series of letters, numbers, and symbols used to send and receive an email [1].
- Passwords - secret words or combinations of letters or numbers, used for communicating with another person or with a computer to prove who you are [2].
- Usernames - a unique sequence of characters used by a person with access to a computer, network or online service [5].
- Phone numbers - numbers assigned to a telephone line for a specific phones or set of phones (as for a residence) that are used to call those phones [4].
- Additional - data types specified and implemented by the individual users.

1.3 Document structure

This document is divided into seven chapters with following focus. First Chapter focuses on the presentation of the problem domain and scope, introduction to the

document structure. The second chapter researches the topic and design of the program. Third chapter describes the used technologies and required prerequisites. Chapter number four is a instruction of the program usage. Fifth one elaborates on the program internal structure. Sixth chapter presents the testing methodology and results and the final seventh presents the final remarks and conclusions for the future of the project.

Chapter 2

Problem analysis

In this chapter an analysis of solutions tackling similar problems is performed. It looks into methods used in them and checks their validity in the project scope.

2.1 Existing solutions

There are not many widely available solutions that are capable of performing the job presented in the thesis introduction. However the scope of the project shares similarities with Data Leakage Prevention Systems which focus on analysis of the content of confidential data and the surrounding context in order to prevent unwanted disclosures of information. Those programs usually use up to three techniques to monitore the sensitive information - regular expressions (regex), data fingerprinting and statistical analysis [13]. An example of DLP software known to implement such features is MyDLP which has a open source community edition as well as closed source enterprise version [6]. There are also other solutions that probably use the aforementioned approach like Symantec Data Loss Prevention, Trustwave Data Loss Prevention, McAfee Total Protection for Data Loss Prevention and Check Point Data Loss Prevention. Similar techniques could be adapted for the purposes of this thesis.

Regular expressions are an abstraction of keyword search that enables the identification of text using a pattern instead of an exact string. They are a tool frequently used for parsing users input and capturing parts of strings [15]. Regexes

are used in Data Leakage Prevention Systems for detecting data like credit card numbers and social security mumbers [13] which belong to the scope of this thesis. It might be possible to extend this approach in to other categories of information covered by the designed program.

Regular expressions are however known to have high false positives rates [13]. It might be advantageous for the purposes of this project to take under consideration additional properties of the considered data types. It is possible for the personal information issued by some entities to implement a check sum algorithm used for validation of authenticity. Implementing an adjustable additional check of the data discovered by regular expressions could possibly reduce the amount of false positives. Such feature could be utilised by individual user to further enhance accuracy in any case where identifing additional patterns, that escape regular expression domain is possible.

While dictionary search does not seem to be a common part of Data Leakage Prevention Systems it might prove useful in achieving the goal of this thesis. The natural language messages processed by the developed program may contain keywords indicating presence of the desired data, which might have not been detected by the initial regex search. Implementing a dictionary search which focuses on words like "password", "login" etc. could potentially increase the reliability of the solution.

2.2 Designing search conditions

It is important to consider how to properly approach designing regualr expressions and additional checks for some data types. For example it is relatively easy to implement search conditions for a social security number due to its specific structure. The scope can be additionally narrowed down if it implements some sort of check sum algorithm. Same can be said about email addresses, ip addersses, MAC addresses, ID card numbers, domain names and phone numbers.

However some data types might prove more difficult. A password or login can take almost any form and be indistinguishable from regular words. Searching for any string of characters can yield results practically imposible to analyse due to the quantity of false positives. However it can be assumed that a password policy

requires it to consist of arbitrary minimum number of characters, contain capital and small letters and numericals. Implementing a reugualr expression and additional check function under those conditions could greatly decrease the percentage of false positive rasults, making it valuable for manual inspection and validation.

The undesired consequence of setting such constrains can be omitting of the positive results in the search process. This could possibly be partialy mitigated by dictionary search for the same data type with proper keywords like "password", "pass" etc.

2.3 Modularity and Customisation

Each use case of the designed system might differ depending on the user, enviroment and multitude of other conditions. Some clients of the program might be interested in searching for a subset of data types implemented in the program. It is also possible that an unanticipated in the design information type might be of a particular interest for a user.

In order to mitigate those problems a modular approach to the design of program could be taken. Its main goal would be to allow specifing which of the stock implemented data types to search for as well as simplifying the augmentation of the program's scope.

It is also crucial to take under consideration the fact that the structure of some data types can differ significantly between uses. An example of this is a personal identification number which may vary depending on issuing authority. Giving the user an easily accessible possibility to adjust the regular expressions used in the searching process as well as additional check functions may increase the use cases of the solutions.

2.4 Result presentation

While the center of this thesis is a program which focuses on finding some sensitive data, it might be profitable to put a special emphasis on the proper presentation of the results. Information discovered by the solution might be riddled

with false positives. In such case giving the user a possibility to easily analyse the message and conversation context of found data could increase the usefulness of the tool.

2.5 Enviroment independance

The typical user of the developed program might need to run it on multiple different operating systems (OS). Administrators might require the program to run on Windows operating system while IT security specialists could possibly want to deploy it on a Kali Linux which is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Developing a program capable of running regardless of the enviroment in which it is used would possibly allow for a bigger user base.

2.6 Input variability

The program is supposed to be capable of operating on many different types of input like emails, Facebook messages, Slack messages etc. In order to allow for that it may be necessary to design an arbitrary input format. A user would have to be required to transform initial data in its possession to fit this predefined standard.

Chapter 3

Requirements and tools

This chapter tackles tools and technologies used during development. It also presents the functional and non-functional requirements of the program.

3.1 Tools

In order to achieve the goal of this thesis following tools were used:

- Visual Studio Code - a source-code editor developed by Microsoft for Windows, Linux and macOS. Its source code is free, open source and released under the permissive MIT License. It was used with a Python plugin which provides a rich support for the Python language, including features such as IntelliSense, linting, debugging, code navigation, code formatting, Jupyter notebook support, refactoring, variable explorer, test explorer, snippets, and more [8][17].
- Qt Designer - tool for designing and building graphical user interfaces (GUI) with Qt Widgets. Allows for composing and customization of windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and testing of them using different styles and resolutions [10].

3.2 Technologies

Entire project is developed with use of following technologies:

- Python 3.7.3 - an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programing. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms [21]. There are multiple arguments that support choosing it as primary development language for the project. It has implemented modules for regular expression search and graphical user interface. The scriptive nature of it makes it easy to create programs that are capable of runing in any operating system that has Pyhton interpreter installed. It is also familiar to the developer of the program.

- JSON - a lightweight data-interchange format. It is easy for humans to read and write and straightforward for machines to parse and generate. Based on a subset of the JavaScript Programing Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [7].

3.3 Requirements

The program designed in this thesis is supposed to perform following tasks:

- Perform a regex search on natural language message sets.

- Additionally check data found with regular expressions with false positives check function if one is available.

- Perform dictionary search on natural language message sets.

- Display the found data.

- Allow for displaying of message context of found data.

- Allow for implementation of custom searches.
- Save to and load results from an output file.

The program designed in this thesis is supposed to be capable of runing regardless of an operating system.

Chapter 4

External specification

Following chapter describes which requirements have to be satisfied for the program usage and how to handle it.

4.1 Requirements and installation

The project consists of files "program.py" contained in the main program directory and "codeLib.py", "core.py", "regexLib.json" stored in "src" subdirectory. All of them are required for the proper operation of the solution.

For the program to work correctly, some software requirements have to be meet. It was developed with Python and requires its interpreter in version at least 3.7.3 to be installed. Additionally a package called PyQt5 has to be in the Python interpreter library. It can be obtained with package-management system called pip. Example of installation script that fulfill the above requirements can be found in figure 4.1 for Windows and 4.2 for Linux. The rest of the Python packages used in the project are usually contained within standard Python installation. In individual cases additional installation may be required.

The solution was not tested from the hardware requirements point of view

```
1   python -m pip install pyqt5
```

Figure 4.1: Windows installation script

```
1 #!/bin/bash
2 sudo pip3 install PyQt5
```

Figure 4.2: Linux installation script

however during development it was deployed on a computer with following specifications:

- Intel Core i5-6300HQ CPU 2.30 GHz.
- 8GBytes DDR4 RAM.
- Samsung SSD 860 EVO M.2 500GB.

4.2 User manual and usage examples

Program described in this section is designed to perform a regular expression search on a dataset of messages supplied form a input file. Found results can be additionally checked with a functions implemented in Python language. A dictionary search which checks if a string matched with regular expression belongs to a provided wordlist can also be performed.

The obtained results are saved to the output file and presented in the GUI.

Following search types can be carried out:

- IP v4
- IP v6
- Social Sec no
- ID no.
- MAC
- Domains
- Emails

- Passwords
- Logins
- Phone no.
- Dictionary
- Additional

The program requires following inputs:

- Input path - specifies location of the input file. It is obligatory for the program execution.
- Output path - provides the output location used for saving the results of program execution. If left empty, the program appends the input path with "_output" suffix.
- Dictionary path - points to the dictionary file. The proper format of a dictionary is a text file filled with all lower case words, separated with new line characters. This path is required only in case of performing a dictionary search.

The configuration of the program is stored in files regexLib.json and codeLib.py.

The program requires preparation of the input data in a JSON file containing an array of objects composed of two key-value pairs. Key "id" needs to be supplied with unique id value for each object. Key "content" is supposed to be paired with message text, this value will be processed in the search operation of the program. Example proper input file is presented in figure 4.3.

Configuration of the program is done partialy with two files contained in its file structure. The "regexLib.json" contains the regular expression of a given data type and a optional name of the method used to perform additional validation. This JSON file is structured in a following maner. It contains three objects marked with arbitrary keys: "main", "additional", "dictionary". These represent categories used for different types of search and contain objects consisting of two key-string pairs. The "regex" key represents regular expression utilised in the primary search

```
{
    messages:
    [
        {
            "id":"<unique_id>",
            "content":"<message_content>"
        },...
    ]
}
```

Figure 4.3: Format of the input JSON file

and "code" contains name of the method used for additional validity check. The "code" key can be paired with an empty string if no additional check is required.

The "main" category contains data types marked by following keys:

- "ipv4" - IP version 4.
- "ipv6" - IP version 6.
- "socialsec" - personal identity numbers.
- "id_number" - ID card numbers.
- "mac" - MAC addresses.
- "domain" - Domain names.
- "email" - Email addresses.
- "password" - Passwords.
- "login" - Usernames.
- "phone_number" - Phone numbers.

The "additional" object can be filled with any data type desired by the user as long as it follows convention from the main category. The keys of each of those objects can have any value and represent the names of categories specified by the customers.

```
{
    "main":{
          "ip_v4": {
             "regex":"
                (?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
                \\.)
                {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
                ",
             "code":""
          },
          ...
    },
    "additional":{
    },
    "dictionary":{
          "dictionary": {
             "regex": "[0-9a-zA-Z\u0142\u0119]{3,8}"
          }
    }
}
```

Figure 4.4:

Finally the "dictionary" object contains regular expressions of words that are supposed to be matched with dictionary search. Each object in this category is composed of a single key-value pair. Its key is "regex" and value is the regular expression used in the program.

When creating regular expressions for the "regexLib.json" files it is important to remember the restrictions of the JSON format. Some characters in it might require escaping or representing in UTF-8 notation.

An abbreviation of a example "regexLib.json" file can be found in figure 4.4 and its full version is available the Listings part of the appendix.

The "codeLib.py" file is intended to contain methods for additional validation. The methods are supposed to be written in Python 3, accepting two parameters and returning either "True" if the validation was succesfull or "False" if not. First parameter is a string containing the data discovered by the regular expression and the second is the full content of the message in which it was found.

```
C:\Projects\Thesis\Code> python .\program.py
```

Figure 4.5: Command for starting the program

The example "codeLib.py" file can be found in the Listings part of the appendix.

Starting of the program can be done by runing the "program.py" file with Python interpreter. The proper command might differ depending on the operating system. Example commands run on a Windows OS from the project directory can be found in figure 4.5.

The main window of the program consists of two tabs. The settings tab allows to setup the search criteria and perform the desired job. It features three text inputs that allow for specifing files used during program execution as well as checkboxes for selecting the desired search types. Consequent to specifing the desired data types and required file paths the user can start and stop the search process with buttons labeled respectively "Start" and "Stop". This view can be found in figure 4.6.

Upon completion of the program work results are presented in the "Results" tab. It consists of three columns used for navigation of the results:

- Category - allows for specification of the search category to browse. Clicking on a given category loads the results column.

- Results - allows to browse the found results in a given category. Clicking on a given result loads the occurance column.

- Occurance - allows to browse the ids of messages in which the result was found. Clicking on a given id opens the preview window with the content of the specified message.

An exemplary results view is presented in figure 4.7.

The goal of the preview window is to inspect the context of a found information. It consists of a central part displaying the content of a message and "Previous" and "Next" buttons used for navigation of the messages stored in the input file, relative to the currently presented message. Preview window can be found in figure 4.8.

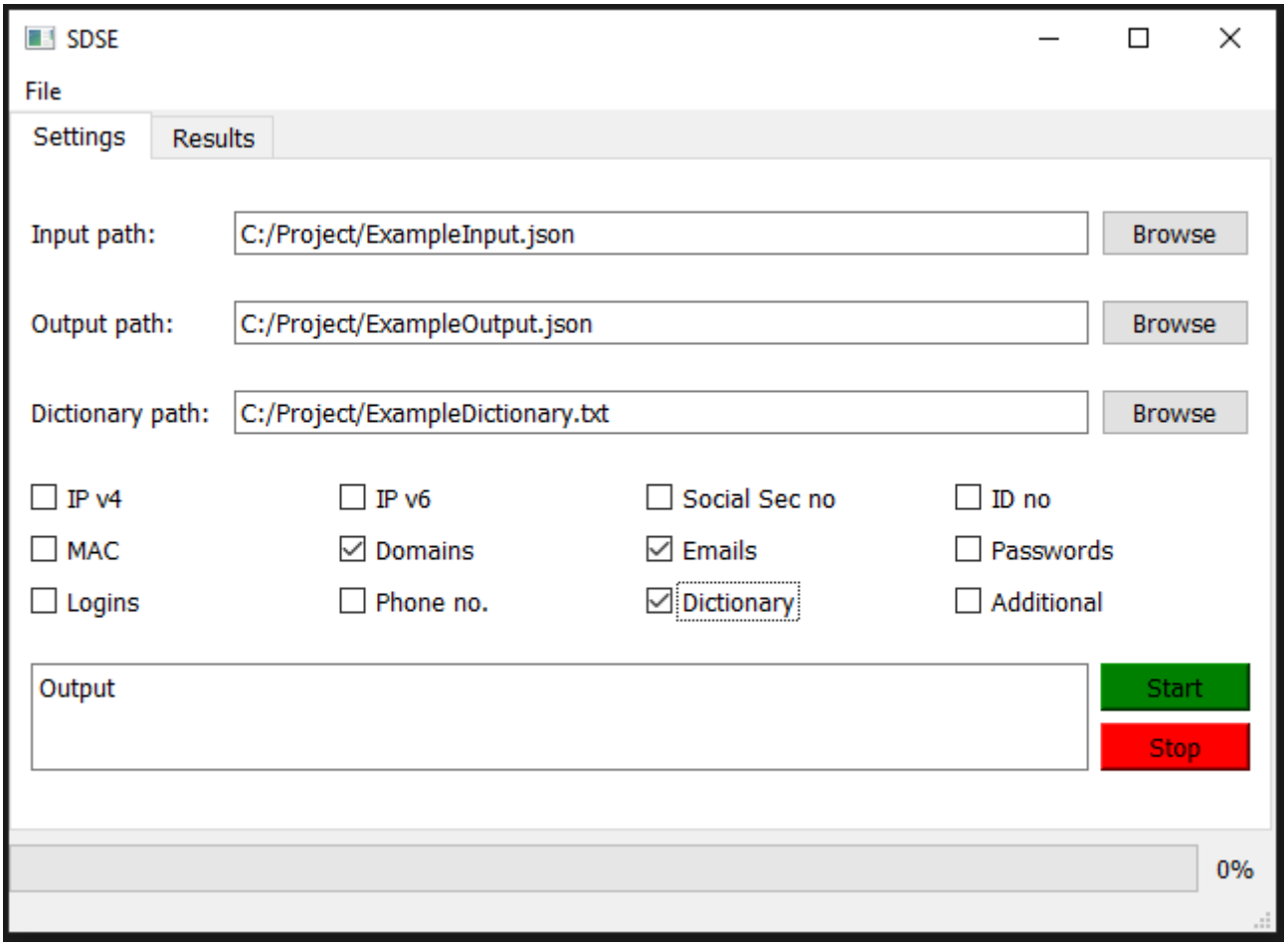

Figure 4.6: Settings view

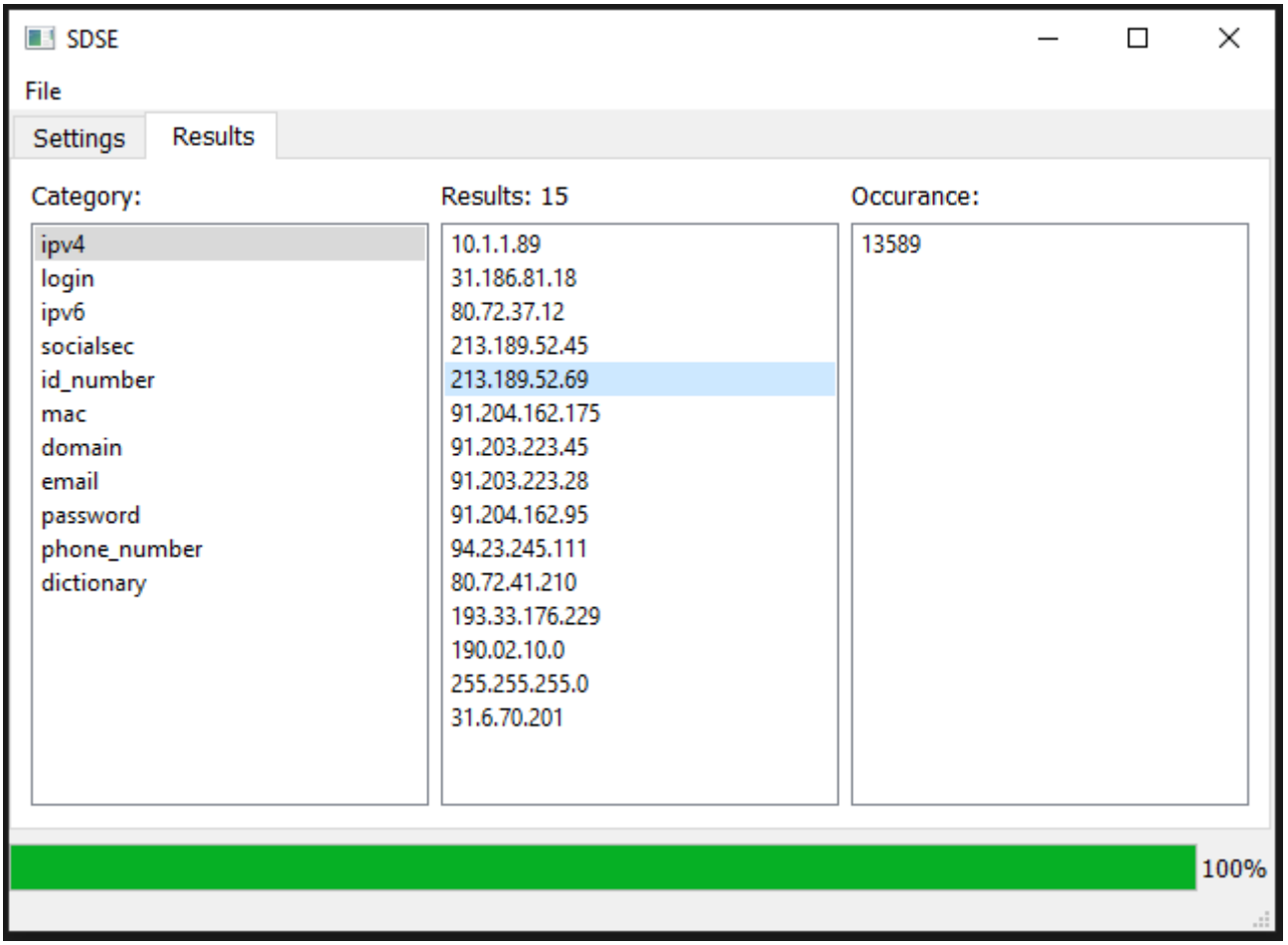

Figure 4.7: Results view

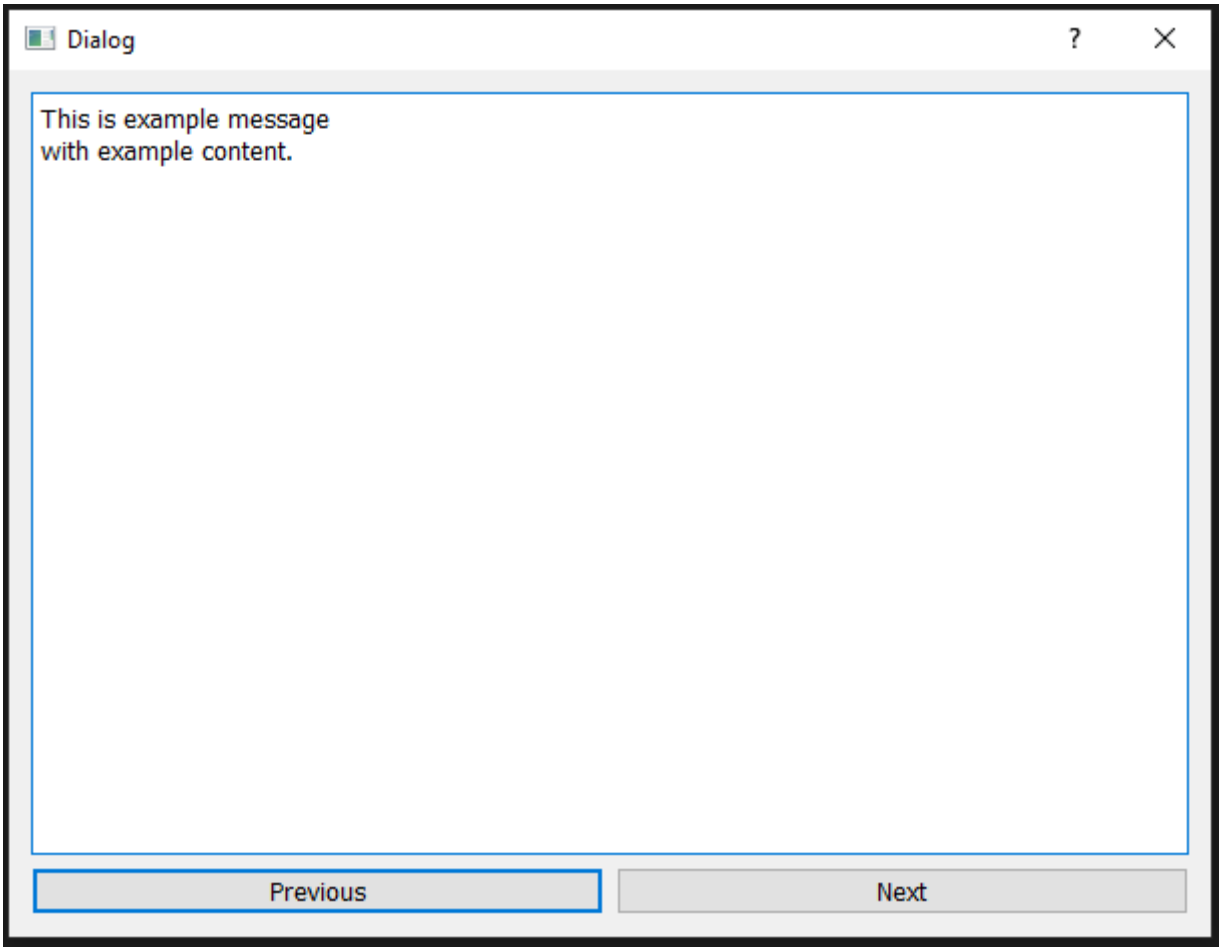

Figure 4.8: Message preview

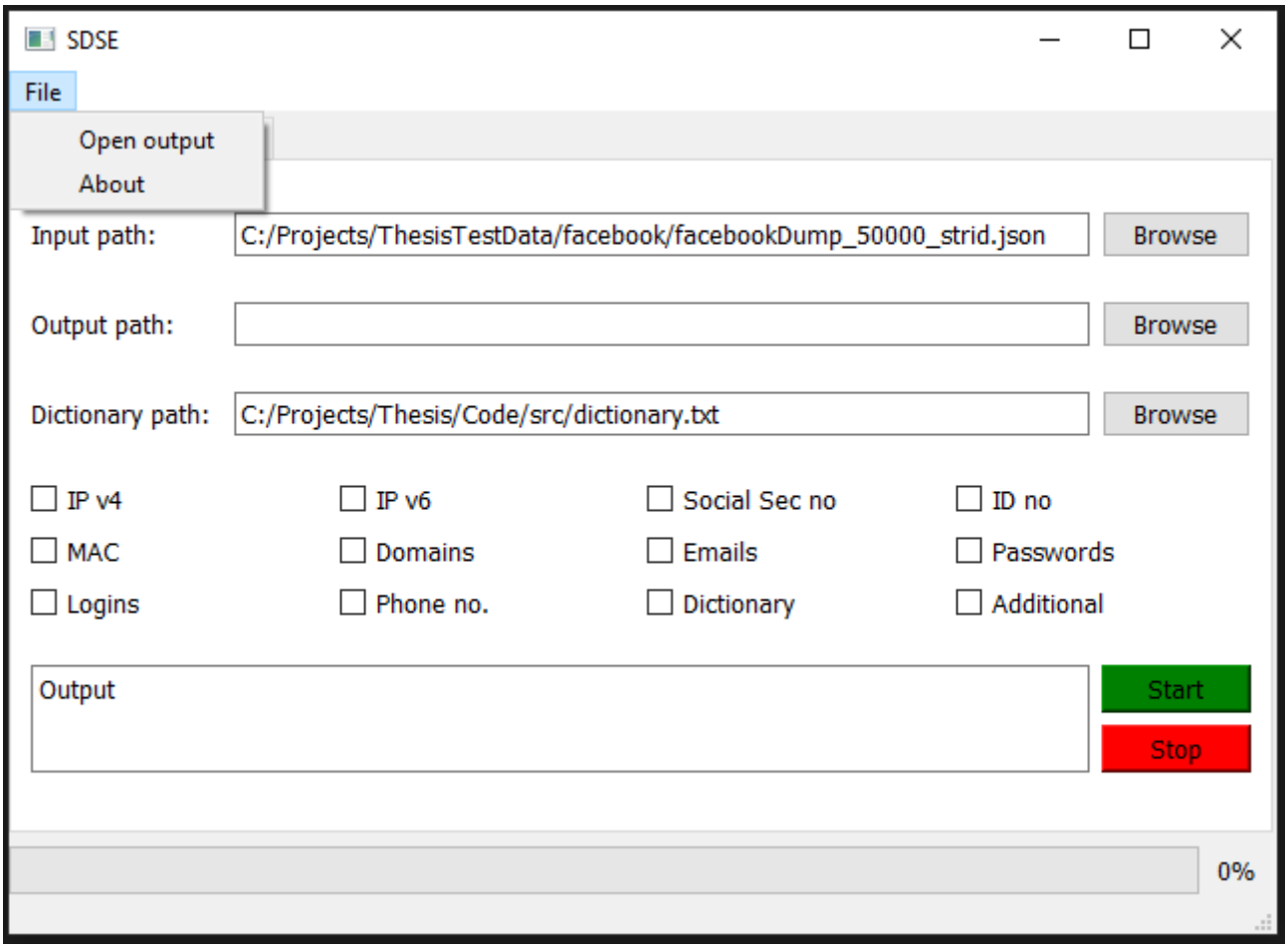

Figure 4.9: Loading output file

It is also possible to load a previously generated output file with the "Open output" option from the "File" menu located in upper left corner of the main window as presented in figure 4.9.

The output of the program is saved to a JSON formated file and takes a form presented in figure 4.10. It consists of three nestings of objects. First layer represents desired search categories. Each of objects in this layer contains number of found occurances marked with the "no" tag and the "results" contains each result. They are stored in separate tags and each additionally contains a list of unique ID's of messages pointing to each occurance.

```
{
   "example": {
         "no": 1,
         "results": {
            "example_finding": {
               "occurances": ["2628", "25920", "25944",
                  "43187", "43187"]
            }
         }
   }, "ipv4": {
         "no": 15,
         "results": {
            "10.1.1.89": {
               "occurances": ["5566", "10568"]
            },
            "31.186.81.18": {
               "occurances": ["13582"]
            }
         }
   }
}
```

Figure 4.10: Example output file

4.3 Security

While the focus of the program is providing additional cyber security and data safety, its improper use could result in security issues. The core of the solution implements dynamic importing of Python code which under some conditions could be used with malicious intent for purposes like privilege escalation. It is important to use the program responsibly. Proper approach should contain at least restraining from executing the program as administrative users. It is additionally recomended to be aware of the functions stored in src/codeLib.py and of the results of their execution. Alternatively the solution could be run in a sandboxed or containerised enviroment without root privelages like a properly configured Docker virtual machine.

Chapter 5

Internal specification

This chapter presents the internal structure of the program. It focuses on the system architecture and used algorithms.

5.1 System architecture

The system consists of two parts, separated into two files, one responsible for graphical user interface and other for performing the goal of the program. The GUI is implemented in two classes called "Ui_window" used for the main window of the program and "Ui_PreviewDialog" responsible for displaying the message context preview. The logic of the system is implemented in classes "Parser" and "Core". The first one is used to retrive individual messages from the input file while the second processes them in search desired data. Additionally the "Signal" class is used to pass messages from the "Core" to the GUI classes. Unified Modeling Language (UML) diagram presenting the individual classes and connections between them can be found in the figure 5.1.

Custom implemented classes utilise following Python packages to perform their tasks:

- PyQt5 - used in GUI implementation.
- JSON - utilised for data input and output.
- os - used for enviroment related actions like proper joining of paths.

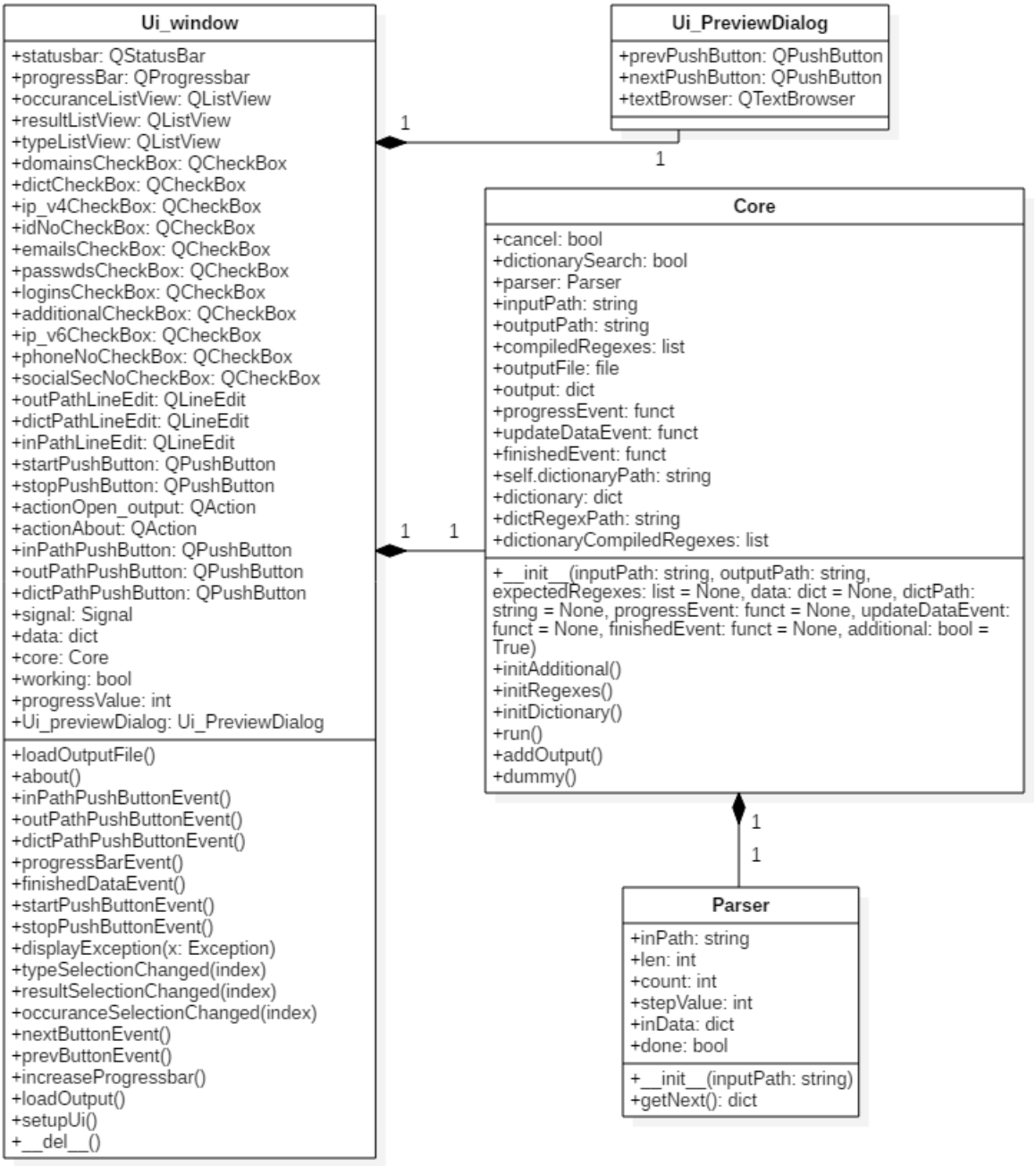

Figure 5.1: UML class diagram

- traceback - required for proper error presentation.
- importlib - allows for dynamic loading of the additional check functions.
- threading - implements Python multi-thread operations.
- re - package for working with Python regular expressions.

5.2 Algorithms

The algorithm taking on the main workload of the solution is a regular expression search. In it each character in the text to be searched is examined in sequence against a list of all possible current characters. During this examination a new list of all possible next characters is built. When the end of the current list is reached, the new list becomes the current list, the next character is obtained, and the process continues. This algorithm continually takes the left derivative of the given regular expression with respect to the text to be searched. The parallel nature of this algorithm makes it extremely fast [20]. This solution utilises the Python module that provides regular expression matching operations.

Chapter 6

Verification and validation

This chapter describes testing's methodology of the solution and its results.

6.1 Data source

Data used in the testing of the program was sourced from a real life Facebook account. The history of communication through the platform was downloaded and adapted to the input format with a python script. The messages were divided into batches of sizes 1000, 2000, 5000, 10000, 20000, 50000, 100000, 227360. The script used for the preparation of data can be found in the Listings section of the appendix. The owner of the data was of a Polish nationality which had to be taken under consideration during the program execution. Due to the personal nature of the data set it had to be redacted and is not a part of this thesis.

6.2 Testing methodology

The entirety of testing was done manualy on a single data set. The effectiveness was tested on a 50000 message set and the efficiency tests were performed on a sample of 227360.

While a context analysis was performed to check if found results are not false positives, it is uncertain if and how many positive results were omitted due to the data format unforeseen in regular expression implementation.

Following regexes were used during the testing:

- IP v4 - "(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"
- IP v6 - "(?:[0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}"
- Social Sec no. - "(?:\s|\A)[0-9]{11}(?:\s|\Z)"
- ID no. - "[A-Za-z]{3}[ ]*[0-9]{6}"
- MAC - "([0-9A-Fa-f]2[:-]){5}([0-9A-Fa-f]{2})"
- Domains - "[a-zA-Z0-9][a-zA-Z0-9-]{1,61}[a-zA-Z0-9]\.[a-zA-Z]{2,}"
- Emails - "(?:\s|\A|:|-)([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)"
- Passwords - "(?:\s|\A)\S{8,32}(?:\s|\Z)"
- Logins - "(?:\s|\A)[a-zA-Z0-9]{3,7}(?:\s|\Z)"
- Phone no. - "(?:\s|\A|:|-)(?:[0-9]{2} ?[0-9]{3} ?(?:[0-9]{2} ?){2}|(?:[0-9]{3}[ ]?){2}[0-9]{3})(?:\s|\Z)"
- Dictionary - "[0-9a-zA-Zł]{4,8}"

Their preparation was partialy based on the regional information of the data owner like a common structure of phone numbers, format of the social security number in the country of orgin, etc. Some of them remain true to the international standards like the one used for email addresses. Third group was designed solely from guesses and some vague standards. This set consists of password and login regular expressions.

Additional check algorithms were implemented for following data types:

- Passwords - checks if found word contains a special character, a numerical, a capital letter and a small letter et the same time
- Login - checks if found word contains a letter and a number

```
1 login
2 haslo
3 has\u0142o
4 password
5 pass
6 username
7 email
8 mail
```

Figure 6.1: Testing dictionary

- Social Sec no. - calculates the check number implemented in polish PESEL and checks its validity.
- ID no. - calculates the check number implemented in polish ID number and checks its validity.

The password and login searches are mutually exclusive based on the length of the found string but follow the same check rules.

Dictionary used in search process consisted of eight keywords presented in figure 6.1. Some of the words used are translations of the ones originating in english to polish. Their focus was on discovering additional data in following categories: Emails, Logins, Passwords.

The additional search was not tested outside of a basic check of proper execution because its use is completely dependent on the user.

6.3 Results

The results of running all the search categories on a 50000 messages sample are presented in table 6.1. Additional analysis of the result showed that multiple found domains were actually subparts of a single domain. It also happened that discovered domains were actually parts of an email addersses. In those cases they were rejected as false positives however when they led to an email as well as domain they were considered valid results.

The analysis of the dictionary keyword search allowed to identify data in some categories. The results can be found in table 6.2

Table 6.1: Results of the execution on test data

Category	Found	False positives	Positives
IP v4	15	1	14
IP v6	0	0	0
Social Sec no.	1	0	1
ID no.	6	2	4
MAC	0	0	0
Domains	516	58	458
Emails	25	0	25
Passwords	105	100	5
Logins	144	143	1
Phone no.	41	0	41

Table 6.2: Dictionary search discoveries

Category	Found	False positives	Positives
Emails	34	29	5
Passwords	33	18	15
Logins	5	3	2

Efficiency of additional checks is presented in table 6.3. Due to the large amount of potential passwords and logins found in the unchecked search, the number of false positives is unknown. Manual context check of this amount of data proved impossible in this thesis.

The timespan of the program execution on a file of size 18400 KB containing 227360 messages can be found in table 6.4. The measurements do not include the timespan of data loading. Evaluations were performed multiple times. The presented results represent the average of the outcomes.

Table 6.3: Influence of additional check

	Additional check			No check		
Category	Found	False positives	Positives	Found	False positives	Positives
Social Sec no.	1	0	1	2	1	1
ID no.	6	2	4	18	16	4
Passwords	105	100	5	25748	unk.	unk.
Logins	144	143	1	15634	unk.	unk.

Table 6.4: Timespan of execution

Category	Time in seconds
IP v4	1.3804
IP v6	1.0012
Social Sec no.	0.8272
ID no.	0.9645
MAC	0.9579
Domains	1.2489
Emails	0.9824
Passwords	1.3278
Logins	1.5423
Phone no.	1.0093
Dictionary	1.2872

Chapter 7

Conclusions

This chapter is an analysis of the solution, development and testing process. It states the possible future development goals and conclusions on the topic.

7.1 Additional tests

The interaction of the users with the solution was not tested during the development process. At this time it is uncertain how the necessity of preparing the data will influence the usefulness of the program. Additional tests should be conducted to analyse how the users interact with the program to conduct if it is usable.

In order to definitely conclude how accurate the search settings are a sample data set should be prepared. In such test sample all quantities of different information types as well as their formats should be known. Such set would allow to analyse how narrowing down the search scope influences the results.

Additionally the solution was tested only on a data set from a single source. Multiple source tests should be conducted to check how different cases impact the usefulness of solution.

7.2 Additional checks

The possibility of performing additional checks can greatly decrease the amount of false positives. Its main limitation is the need to identify a pattern or validation

function specific to the data catogory. In the testing this is especially apparent in the search of logins and passwords. Applying the common practices used during their creation significantly improves the output of the program. A skilled user could analyse the enviroment of the data source to find additional patterns specific to each use. An example of such action could be identifing an internal company password policy and applying it in search. It is also possible to apply more and more precise constrains on the search patterns until reaching an amount of results that is feasible to manual processing.

It is however worth mentioning that a dictionary search of keywords related to passwords and logins is as or more effective than regular expression check of the data and requires relatively less preparation to work. Running a keyword search first and implementing a regex search after insufficient valuable results were discovered might be a proper approach.

7.3 Future development

The search performed on a dataset of 227360 messages is done in a resonable time. In the future development splitting of the search to multiple threads, one for each category can be implemented in order to more efficiently hadle working on extremely large imputs.

It became apparent during testing that in some cases a single result can present as multiple findings. A feature could be implemented to check for reoccuurrence of a found data to improve results presentation. The processing of the output could also be augmented with a classification option that would assign priority to some results based on set weights, results of the additional check function and other factors. For example a dictionary check for known usernames could be compared to the found passwords candidates and used to classify the probability of false positive occurance.

Currently the preview functionality used for displaying of found data and message context does not allow to perform any changes on the results. A possibility of removing false positive results and performing changes on the found snippets of text could greatly improve the usefulness of the program in its working scenarios. This would require implementation of saving the work done to the output file.

In a case where adapting data to the format required for the program proves to be a significant obstruction for the program users, a modular input adapter capable of accepting dataset common in real use cases could be developed. An example of that could be a possibility to read mbox files used to archive emails or loading dumps of slack messages.

The use of a JSON file to store the regular expressions might be uncomfortable for the users due to the need of escaping characters. This problem could be amended by implementing a different storing mechanism or a tool to manipulate the regexes in their natural form and writting them in the transformed form to the file.

7.4 Calibration

The process of calibration of the program may be complicated and dependent on many factors. It might be useful to develop a calibration framework and create a manual for development of search constrains.

It is also possible that distributing the project as an open source solution might encurage the community to contribute calibration rules and procedures for multiple unforeseen use contexts. Such approach might greatly increase the usefulness of the product and its value.

Bibliography

[1] Definition of a email address in cambridge english dictionatry. https://dictionary.cambridge.org/dictionary/english/email-address. [access date: 2019-12-23].

[2] Definition of a password in cambridge english dictionatry. https://dictionary.cambridge.org/pl/dictionary/english/password. [access date: 2019-12-23].

[3] Definition of id by merriam-webster. https://www.merriam-webster.com/dictionary/id. [access date: 2019-12-23].

[4] Definition of telephone number by merriam-webster. https://www.merriam-webster.com/dictionary/telephone%20number. [access date: 2019-12-23].

[5] Definition of username at dictionary.com. https://www.dictionary.com/browse/username. [access date: 2019-12-23].

[6] Features: Mydlp enterprise edition. https://mydlp.com/features/. [access date: 2020-01-17].

[7] Json. https://www.json.org/json-en.html. [access date: 2019-12-23].

[8] License - visual studio code. https://code.visualstudio.com/license. [access date: 2019-12-23].

[9] Media access control address (mac address). https://www.techopedia.com/definition/5301/media-access-control-address-mac-address. [access date: 2019-12-23].

[10] Qt designer manual. `https://doc.qt.io/qt-5/qtdesigner-manual.html`. [access date: 2019-12-23].

[11] What is a domain name? `https://www.website.com/beginnerguide/domainnames/8/1/What-is-a-domain-name?.ws`. [access date: 2019-12-23].

[12] Internet protocol. *DARPA Internet Program Protocol Specification*, 1981.

[13] Sultan Alneyadi, Elankayer Sithirasenan, and Vallipuram Muthukkumarasamy. A survey on data leakage prevention systems. *Journal of Network and Computer Applications*, 62:137–152, 2016.

[14] Rafay Baloch. *Ethical Hacking and Penetration Testing Guide*. Auerbach Publications, 2014.

[15] Kathryn T. Stolee Carl Chapman. Exploring regular expression usage and context in python. In *25th International Symposium on Software Testing and Analysis*, pages 282–293, 2016.

[16] Rajesh Kumar Goutam. Importance of cyber security. *International Journal of Computer Applications*, 111(7):4, 2016.

[17] Frederic Lardinois. Microsoft launches visual studio code, a free cross-platform code editor for os x, linux and windows. `https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-c` [access date: 2019-12-23].

[18] R. Hinden S. Deering. Ip version 6 addressing architecture. *The Internet Society*, 1998.

[19] R. Hinden S. Deering. Internet protocol, version 6 (ipv6) specification. *The Internet Society*, 2017.

[20] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419—422, 1968.

[21] Guido van Rossum and the Python development team. *PythonTutorial Release3.8.1*. Python Software Foundation, 2019.

[22] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer International Publishing, Cham, 2017.

[23] Jim Willeke. National identification number. `https://ldapwiki.com/wiki/National%20Identification%20Number`. [access date: 2019-12-23].

Appendices

List of abbreviations and symbols

OS	operating system
regex	regular expression
GUI	Graphical user interface
UML	Unified Modeling Language

Listings

Example of a "regexLib.json" file:

```
{
   "main":{
         "ip_v4": {
            "regex":"
               (?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
               \\.)
               {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
               ",
            "code":""
         },
         "ip_v6": {
            "regex":"(?:[0-9a-fA-F]{1,4}:){7}[0-9a-fA-F
               ]{1,4}",
            "code":""
         },
         "socialsec":{
            "regex":"(?:\\s|\\A)[0-9]{11}(?:\\s|\\Z)"
            ,"code":"pesel"
         },
         "id_number":{
            "regex":"[A-Za-z]{3}[␣]*[0-9]{6}",
            "code":"idnum"
         },
```

```
19          "mac":{
20             "regex":"([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f
                  ]{2})",
21             "code":""
22          },
23          "domain":{
24             "regex":"[a-zA-Z0-9][a-zA-Z0-9-]{1,61}[a-zA-
                  Z0-9]\\.[a-zA-Z]{2,}",
25             "code":""
26          },
27          "email":{
28             "regex":"(?:\\s|\\A|:|-)([a-zA-Z0-9_.+-]+@[a
                  -zA-Z0-9-]+\\.[a-zA-Z0-9-.]+)",
29             "code":""
30          },
31          "password":{
32             "regex":"(?:\\s|\\A)\\S{8,32}(?:\\s|\\Z)",
33             "code":"passwd"
34          },
35          "login":{
36             "regex":"",
37             "code":""
38          },
39          "phone_number":{
40             "regex":"(?:\\s|\\A|:|-)(?:[0-9]{2}␣
                  ?[0-9]{3}␣?(?:[0-9]{2}␣?){2}|(?:[0-9]{3}[
                  ␣]?){2}[0-9]{3})(?:\\s|\\Z)",
41             "code":""
42          }
43    },
44    "additional":{
45          "l":{
46             "regex": "[\u0142]{1}",
```

```
47              "code": ""
48          },
49          "\u0119":{
50              "regex": "\u0119",
51              "code": ""
52          }
53    },
54    "dictionary":{
55          "dictionary": {
56              "regex": "[0-9a-zA-Z\u0142\u0119]{3,8}"
57          }
58    }
59 }
```

Example of a "codeLib.py" file:

```
1 def pesel(snip, content):
2     snip = snip[1:-1]
3     weight = [9, 7, 3, 1, 9, 7, 3, 1, 9, 7]
4     checksum = (int(snip[0]) * weight[0]) + (int(snip[1])
          * weight[1]) + (int(snip[2]) * weight[2]) + (int(
        snip[3]) * weight[3])+ (int(snip[4]) * weight[4])
        + \
5         (int(snip[5]) * weight[5]) + (int(snip[6]) *
            weight[6]) + (int(snip[7]) * weight[7]) + (int
            (snip[8]) * weight[8]) + (int(snip[9]) *
            weight[9])
6
7     if checksum%10 == int(snip[-1]):
8         return True
9     else:
10        return False
11
12 def idnum(snip, content):
```

```
13    weight = [7, 3, 1, 0, 7, 3, 1, 7, 3]
14    key = {"A": 10, "B": 11, "C": 12, "D": 13, "E": 14, "
         F": 15, "G": 16, "H": 17, "I": 18, "J": 19, "K":
         20, "L": 21, "M": 22, \
15        "N": 23, "O": 24, "P": 25, "Q": 26, "R": 27, "S":
              28, "T": 29, "U": 30, "V": 31, "W": 32, "X":
             33, "Y": 34, "Z": 35}
16
17    snip_ready = snip.replace("␣", "").upper()
18    checksum = (key[snip_ready[0]] * weight[0]) + (key[
         snip_ready[1]] * weight[1]) + (key[snip_ready[2]]
         * weight[2]) + (int(snip_ready[3]) * weight[3]) +
         \
19        (int(snip_ready[4]) * weight[4]) + (int(
             snip_ready[5]) * weight[5]) + (int(snip_ready
             [6]) * weight[6]) + (int(snip_ready[7]) *
             weight[7]) + (int(snip_ready[8]) * weight[8])
20
21    if checksum%10 == int(snip_ready[3]):
22        return True
23    else:
24        return False
25
26 def passwd(snip, content):
27    specialChars = "!@#$%^&*()_+-={}[];:\"'\\|<>?,./"
28    nums = "0123456789"
29    uppers = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30    lowers = "abcdefghijklmnopqrstuvwxyz"
31    containsSpecial = False
32    containsNum = False
33    containsChars = False
34    containsLow = False
35    containsUpper = False
```

```
36
37      for num in nums:
38          if num in snip:
39              containsNum = True
40              break
41
42      if not containsNum:
43          return False
44
45      for char in specialChars:
46          if char in snip:
47              containsSpecial = True
48              break
49
50      if not containsSpecial:
51          return False
52
53      for low in lowers:
54          if low in snip:
55              containsLow = True
56              break
57
58      if not containsLow:
59          return False
60
61      for up in uppers:
62          if up in snip:
63              containsUpper = True
64              break
65
66      if not containsUpper:
67          return False
68      else:
```

```
69          return True
```

Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (LaTeX source files and final `pdf` file),
- source code of the application

List of Figures

List of Tables