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, DSc PhD

consultant: Michał Kawulok, PhD

Gliwice, December 2019

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

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

Gliwice, dnia 27 grudnia 2019

. . . .
(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 27 grudnia 2019

. . . .

(podpis promotora)

Contents

1 Introduction 1

1.1 Description of the problem 1

1.2 Project scope 2

1.3 Description of chapters 3

2 Problem analysis 5

2.1 Existing solutions 5

2.2 Modularity and Customisation 6

2.3 Result presentation 7

2.4 Enviroment independance 7

2.5 Input variability 7

3 Requirements and tools 9

3.1 Tools 9

3.2 Technologies 9

3.3 Requirements 10

4 External specification 13

4.1 Requirements and installation 13

4.2 User manual and usage examples 14

4.3 Security 18

5 Internal specification 19

5.1 System architecture 19

5.2 Algorithms 20

6 Verification and validation 21

6.1 Data source 21

6.2 Testing methodology 21

6.3 Results 23

7 Conclusions 27

Chapter 1

Introduction

This chapter presents the problem that the project tries to solve and presents 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 word 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 rize of the impact of technology was followed by the rise in the cyber crime and cyber security providers [14].

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 [12]. The reason for that is because the more information you can uncover and analyze, the bigger the chance of finding vulnerable systems or flaws in them. One of the clusters of information in in companies is a communication channel like slack or discord. There are manny situations where employees share information connected to projects and their workplace enviroment. If an attacker was to acces 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 manny legal bodies to responsibly handle people's personal data under a threat of heavy financial penalties [19]. 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 addresses - numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication [16].
- National identification numbers - numbers issued by a Government Entity as a means of providing Identification of their citizens and or Aliens[20]. 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 [8].

- 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[10].

- Email addresses - serieses of letters, numbers, and symbols used to send and receive 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 users.

1.3 Description of chapters

This document is divided into seven chapters with following focus:

- Chapter 1 Introduction - Presentation of the problem domain and scope, introduction of the document structure.

- Chapter 2 Problem analysis - Research of the topic and design of the program.
- Chapter 3 Requirements and tools - Description of the used technologies and required prerequisites.
- Chapter 4 External specification - Instruction of the program usage.
- Chapter 5 Internal specification - Elaboration on the programs internal structure.
- Chapter 6 Verification and validation - Presentation of the testing methodology and results.
- Chapter 7 Conclusions - Final remarks and conclusions for the future of the project.

Chapter 2

Problem analysis

2.1 Existing solutions

There are not manny 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 Lekage 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 (regexes), data fingerprinting and statistical analysis [11]. This approach could be adapted for the purposes of this thesis.

Regular expressions are an abstraction of key-word search that enables the identification of text using apattern instead of an exact string. They are a tool frequently used for parsing users input and capturing parts of strings [13]. Regexes are used in Data Leakage Prevention Systems for detecting data like credit card numbers and social security mumbers [11] which belong to the scope of this thesis. It might be possible to extend this approach in to other categories of data covered by the designed program.

Regular expressions are however known to have high false positives rates [11]. 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 a 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 Modularity and Customisation

Each usecase 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 a easily accessible possibility to adjust the regular expressions used in searching process as well as addictional check functions may increase the use cases of the solutions.

2.3 Result presentation

While the focus of this thesis is a program which focuses on finding some sensitive data, it might be useful to put a special emphasis on the 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.

2.4 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 runing regardless of the enviroment in which it is used would possibly allow for a bigger user base.

2.5 Input variability

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

Chapter 3

Requirements and tools

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. It's source code is free and 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 [15][7].
- 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[9].

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 programming. 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[18]. There are multiple arguments that support choosing it as privary 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.

- JSON - a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming 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 programmers 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[6].

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 a output file.

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

Chapter 4

External specification

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 program.

For the program to work correctly, some software requirements have to be meet. It was developed with Python and requires it's interpreter in version at least 3.7.3 to be installed. Additionally a package called PyQt5 has to be installed in the Python interpreter library.

Some of those are contained within standard Python installation and the rest can be obtained installed 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 figure 4.2 for Linux.

The solution was not tested from the hardware requirements point of view however during development it was deployed on a computer with following specifications:

- Intel Core i5-6300HQ CPU 2.30 GHz

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

Figure 4.1: Windows installation script

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

Figure 4.2: Linux installation script

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

Figure 4.3: Format of the input JSON file

- 8GBytes DDR4 RAM
- Samsung SSD 860 EVO M.2 500GB

4.2 User manual and usage examples

The program requires preparing of the input data in a JSON file containing a array of objects composed of two key-value pairs. Key "id" needs to be suppied 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 it's file structure. The "regexLib.json" contains the regular expression of a given data type and a optional name of the method used to to perform additional validation. This JSON file is structured in a following maner. It contains three objects marqued 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 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:

- "ip_v4" - IP version 4.
- "ip_v6" - 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.

Finally the "dictionary" object is 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. It's 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 json format might require escaping characters or representing them in UTF-8 notation.

The example "regexLib.json" file can be found in the Listings part of the appendix.

The "codeLib.py" file is intended to contain methods for additional validation. The methods are supposed to be writen in Python 3, accept two parameters and

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

Figure 4.4: Command for starting the program

return 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.

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 4.4.

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 durring program execution.

- 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 txt 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 settings tab allows also to specify what types of data to search for with checkboxes labeled:

- IP v4
- IP v6
- Social Sec no

- ID no.
- MAC
- Domains
- Emails
- Passwords
- Logins
- Phone no.
- Dictionary
- Additional

Uppon 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".

Uppon complition of the programs 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 id's of messages in which the result was found. Clicking on a given id opens the preview window with the content of the specified message.

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.

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.

4.3 Security

While the focus of the program is providing additional cyber security and data safety, it's 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.

Chapter 5

Internal specification

5.1 System architecture

The system consists of two parts, 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 a "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

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 or enviroment related actions like proper joining of paths.
- 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 the regular expression recognition technique 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 [17]. This solution utilises a Python module that provides regular expression matching operations.

Chapter 6

Verification and validation

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 sample and the efficiency tests were performed on 227360 message set.

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

Following regexes were used durring 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 bassed 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
- Social Sec no. - calculates the check number implemented in polish PESEL and checks it's validity.

```
login
haslo
has\u0142o
password
pass
username
email
mail
```

Figure 6.1: Testing dictionary

- ID no. - calculates the check number implemented in polish ID number and checks it's 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 it's 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 a 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 additional data in some categories. The results can be found in table 6.2

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

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 additional discoveries

Category	Additional result
Emails	0
Passwords	4
Logins	3

false positives is unknown. Manual context check of this amount of data proved imposible in this thesis.

The timespan of the program execution on a 227360 samples data set can be found in table 6.4.

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	1.3349
IP v6	0.9224
Social Sec no.	0.8259
ID no.	0.9299
MAC	0.8999
Domains	1.2889
Emails	0.9979
Passwords	1.3460
Logins	1.5800
Phone no.	1.0913
Dictionary	1.3450
All	9.3316

Chapter 7

Conclusions

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

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

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

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

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

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

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

[13] 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.

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

[15] 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].

[16] Name Surname and Name Surname. Internet protocol. *DARPA Internet Program Protocol Specification*, 1981.

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

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

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

[20] 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"
         },
          "mac":{
             "regex":"([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})",
             "code":""
          },
          "domain":{
             "regex":"[a-zA-Z0-9][a-zA-Z0-9-]{1,61}[a-zA-Z0-9]\\.[a-zA-Z]{2,}",
             "code":""
          },
          "email":{
             "regex":"(?:\\s|\\A|:|-)([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+)",
             "code":""
          },
          "password":{
             "regex":"(?:\\s|\\A)\\S{8,32}(?:\\s|\\Z)",
             "code":"passwd"
          },
          "login":{
             "regex":"",
             "code":""
          },
          "phone_number":{
             "regex":"(?:\\s|\\A|:|-)(?:[0-9]{2} ?[0-9]{3} ?(?:[0-9]{2} ?){2}|(?:[0-9]{3}[ ]?){2}[0-9]{3})(?:\\s|\\Z)",
             "code":""
          }
    },
    "additional":{
          "l":{
             "regex": "[\u0142]{1}",
                "code": ""
            },
            "\u0119":{
                "regex": "\u0119",
                "code": ""
            }
    },
    "dictionary":{
            "dictionary": {
                "regex": "[0-9a-zA-Z\u0142\u0119]{3,8}"
            }
    }
}
```

Example of a "codeLib.py" file:

```
def pesel(snip, content):
    snip = snip[1:-1]
    weight = [9, 7, 3, 1, 9, 7, 3, 1, 9, 7]
    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]) + \
        (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])

    if checksum%10 == int(snip[-1]):
        return True
    else:
        return False

def idnum(snip, content):
    weight = [7, 3, 1, 0, 7, 3, 1, 7, 3]
    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, \
        "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}

    snip_ready = snip.replace("␣", "").upper()
    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]) +
       \
        (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])

    if checksum%10 == int(snip_ready[3]):
        return True
    else:
        return False

def passwd(snip, content):
    specialChars = "!@#$%^&*()_+-={}[];:\"'\\|<>?,./"
    nums = "0123456789"
    uppers = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    lowers = "abcdefghijklmnopqrstuvwxyz"
    containsSpecial = False
    containsNum = False
    containsChars = False
    containsLow = False
    containsUpper = False
    for num in nums:
        if num in snip:
            containsNum = True
            break

    if not containsNum:
        return False

    for char in specialChars:
        if char in snip:
            containsSpecial = True
            break

    if not containsSpecial:
        return False

    for low in lowers:
        if low in snip:
            containsLow = True
            break

    if not containsLow:
        return False

    for up in uppers:
        if up in snip:
            containsUpper = True
            break

    if not containsUpper:
        return False
    else:
        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

4.1 Windows installation script 13
4.2 Linux installation script 14
4.3 Format of the input JSON file 14
4.4 Command for starting the program 16

6.1 Testing dictionary 23

List of Tables

6.1 Results of the execution on test data 24
6.2 Dictionary search additional discoveries 24
6.3 Influence of additional check 24
6.4 Timespan of execution 25