Password Policy Crawler

Master-Thesis von Mario Schlipf
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Moritz Horsch

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Kryptographie und Computeralgebra

Password Policy Crawler

Vorgelegte Master-Thesis von Mario Schlipf

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Moritz Horsch

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 15. Dezember 2015

(Mario Schlipf)

Abstract

User accounts at services on the Internet contain many privacy and security sensitive data such as emails, pictures, health records, and bank account details. Despite several security and usability drawbacks, passwords remain the most widely used authentication scheme to protect user accounts against unauthorized access. Usually, users tend to select passwords that they can easily remember. However, these passwords are highly predictable and prone to brute-force and dictionary attacks. Therefore, security experts recommend to use a password generator in order to create a random and cryptographically secure password for a user acccount.

Using existing password generators, users are faced with the issue that the generated password might not be accepted by the service because it does not fulfill the serivce's password requirements. For instance, the password is too short or does not contain a required special character. One possible solution to cope with this problem is to adjust the password until it fulfills the requirements and gets accepted by the service. Another possible solution is to configure the generator with regard to the respective requirements of the particular service. However, this is not possible in all cases because password generators do not provide means for all kinds of requirements. For instance, a service requires the first character of the password to be alphanumeric. Both approaches are highly inconvenient and time-consuming for users so that they avoid using password generators and keep choosing *weak* passwords. Hence, a novel mechanism is required that allows password generators to easily create secure passwords in accordance with a service's password requirements.

To enable password generators to consider the password requirements, a standardized way for expressing password requirements is necessary. The Password Policy Markup Language (PPML) [1] provides a standardized description for password requirements. However, these password policies have to be created manually. This requires a tremendous effort to do this for the huge amount of services present on the Internet. Consequently, a solution to automatically create such password policies is needed.

This thesis introduces the Password Policy Crawler (PPC). The PPC browses a service's website, extracts its password requirements, and creates a corresponding password policy automatically. It uses modern Natural Language Processing technologies to precisely identify and extract the requirements from a service's website. The accuracy of the PPC was evaluated on 200 services. It is shown that the PPC finds the password requirements of 74.5% of all services and successfully extracts them in 91.5%. The PPC is a highly scalable application which is demonstrated in a conducted large-scale creation of password policies for 72,125 services. Additionally, the first password generator is presented that automatically generates secure passwords in accordance with the password requirements of the services. Users only need to provide the URL of a service which eliminates the burden of manually looking up the requirements and configuring the generator. This simplifies the usage of password generators and makes it easier for users to create secure passwords.

Zusammenfassung

Heutzutage speichern Nutzer eine Vielzahl von vertraulichen Daten wie E-Mails, Bilder, Gesundheitsakten und Finanzdaten bei Diensten im Internet. Zum Schutz der Daten vor unberechtigtem Zugriff werden vorwiegend Passwörter eingesetzt. Bei der Wahl von Passwörtern neigen Nutzer dazu, einfache, leicht zu merkende Passwörter zu verwenden. Dies führt jedoch dazu, dass die Passwörter leicht zu erraten sind. Sicherheitsexperten empfehlen daher den Einsatz eines Passwort-Generators, um zufällige und kryptographisch sichere Passwörter für Nutzerkonten zu generieren.

Bei der Verwendung heutiger Passwort-Generatoren existiert jedoch das Problem, dass ein generiertes Passwort unter Umständen nicht von einem Dienst akzeptiert wird. Das Passwort ist beispielsweise zu kurz oder enthält ein nicht erlaubtes Sonderzeichen. Nutzer können dieses Problem auf unterschiedliche Weise lösen. Zum einen kann das generierte Passwort vom Nutzer angepasst werden, bis es den Anforderungen des Dienstes entspricht. Zum anderen kann der Generator für den Dienst konfiguriert werden, sodass dieser Passwörter gemäß den Anforderungen generiert. Dies ist aber nicht immer möglich. Beispielsweise unterstützen existierende Passwort-Generatoren Anforderungen wie etwa, dass das erste Zeichen eines Passworts alphanumerisch sein muss, nicht. Beide Lösungsansätze sind umständlich und zeitaufwändig, wodurch Nutzer Passwort-Generatoren meiden und stattdessen weiterhin *schwache* Passwörter einsetzen. Es bedarf daher einer Lösung, die es Passwort-Generatoren erlaubt, auf einfache Weise Passwörter zu generieren, die den Anforderungen der jeweiligen Dienste genügen.

Für die Entwicklung von Passwort-Generatoren, die automatisch gültige Passwörter generieren, bedarf es einer standardisierten Beschreibungssprache für Passwort-Anforderungen. Die Password Policy Markup Language (PPML) [1] stellt eine solche Beschreibungssprache bereit, erfordert aber die manuelle Erstellung der Password Policies. Jedoch erweist sich die manuelle Erstellung für die Vielzahl an Diensten im Internet als nicht durchführbar.

Diese Thesis präsentiert den Password Policy Crawler (PPC). Der PPC besucht die Webseite eines Dienstes, extrahiert dessen Passwort-Anforderungen und erzeugt automatisch eine entsprechende Password Policy. Durch den Einsatz der Technologie des Natural Language Processing ist der PPC in der Lage, die Passwort-Anforderungen einer Website präzise zu erkennen und zu extrahieren. Die Genauigkeit des PPC wird anhand von 200 Diensten evaluiert. Es wird gezeigt, dass der PPC die Passwort-Anforderungen für 74,5% der Dienste findet und in 91,5% der Fälle erfolgreich extrahiert. Der praktische Einsatz des PPC wird durch das Erstellen von Password Policies für 72.125 Dienste gezeigt. Des Weiteren wird der erste Passwort-Generator vorgestellt, der das automatische Erzeugen von Passwörtern unter Berücksichtigung der Dienst-spezifischen Passwort-Anforderungen erlaubt. Ein Nutzer muss hierfür lediglich die URL des Dienstes angeben, wodurch das manuelle Konfigurieren des Passwort-Generators bzw. die nachträgliche Anpassung des Passworts entfällt. Für Nutzer stellt dies eine wesentliche Vereinfachung in der Benutzung von Passwort-Generatoren dar.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Annotation Engine
AJAX	Asynchronous JavaScript and XML
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CAS	Common Analysis Structure
CC	Content Crawler
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Information Extraction
ISO	International Organization for Standardization
NLP	Natural Language Processing
PDF	Portable Document Format
PPC	Password Policy Crawler
PPE	Password Policy Extractor
PPML	Password Policy Markup Language
SBD	Sentence boundary disambiguation
SSO	Single Sign-On
TAE	Text Annotation Engine
TLD	Top Level Domain
UIM	Unstructured Information Management
UIMA	Unstructured Information Management Architecture
URL	Unified Resource Locator
WWW	World Wide Web
XML	eXtensible Markup Language
XSD	XML Schema Definition

Acknowledgement

Above all, I would like to thank my supervisor, Moritz Horsch, and Prof. Dr. Johannes Buchmann for giving me the opportunity of writing my thesis on this very interesting and challenging topic. In particular, I want to thank Moritz Horsch. I am deeply grateful for his continuous support. His guidance and commitment helped me in all the time of research and writing my thesis. It was a great pleasure to have him as supervisor and work with him in all these months. Furthermore, I want to thank Martin Riedl for his helpfulness and giving constructive input in certain topics of Natural Language Processing.

Finally, I want to thank my family and friends for their support not only during this thesis, but throughout all the years of my studies.

1 Introduction

Internet usage and online applications are experiencing spectacular growth. Online applications and services store user accounts containing a broad spectrum of sensitive data, ranging from personal communication to health records and financial information. In order to protect this data, a secure authentication is necessary to grant access only to authorized users. While there exist multiple approaches for user authentication, passwords remain the most widely used authentication scheme [2], despite several security and usability drawbacks.

To prevent brute force and dictionary attacks, passwords must be strong. Weak passwords undermine the security and put user data at risk [3]. Furthermore, passwords should not be reused [4] and have to be memorized for later use. The reuse of passwords creates another security risk, as an attacker who is able to compromise one service can compromise other services protected by the same password. The increasing number of accounts [5] users have to deal with makes it nearly impossible to rememeber a unique password for each service. To overcome this problem, users can use password managers that allow the secure storage of passwords. Consequently, users are able to choose unique passwords for each service without the need to memorize them. Moreover, password generators [6, 7, 8, 9, 10] assist users in the creation of random and secure passwords. However, because different service providers apply different requirements such as the allowed characters or the length of the password, the generated passwords might not be accepted by the service. To cope with this problem, users choose unnecessary weak passwords only matching the minimum requirements. For services with more complex requirements, users use simple tricks as workaround, for example adding the number 1 to the end of their default password to fulfill the requirements. In order to generate secure and valid passwords, users must manually find out a service's requirements and configure the password generator, which is time consuming and error-prone. Unusual password requirements might not even be configurable. The generation of secure passwords that comply with a service's requirements is therefore still an open issue.

To address the issue of different password requirements, the Password Policy Markup Language (PPML) [1] provides a standardized description of a service's password requirements. Based on such well-defined descriptions, so-called password policies, password requirements can be expressed in a machine-readable format. Password generators can use this information to automatically generate secure passwords for a service in accordance with its requirements. This allows an easy-to-use solution for generating secure and unique passwords for each used service. Yet PPML only provides the description of password requirements, the policies still need to be created. Currently, these policies do not exist and it is unlikely that a large fraction of internet services will provide them on their own initiative. A community-based approach can allow users to create such policies and submit them to a central entity for use by others. While this might work for popular services, it will not scale for the huge amount of services present on the internet.

This thesis introduces the Password Policy Crawler (PPC), a solution to create password policies automatically. The PPC is a software application that extracts the password requirements from a service's website and generates the corresponding password policy automatically. Because the password requirements are presented to the user while creating an account for the service, they can generally be found on the signup page. Therefore, the PPC first finds the signup page of the service. Second, the content of the signup page is analyzed. The PPC extracts requirements from texts present on the signup page (e.g. "Use at least one uppercase letter") as well as from the password input field that may contain information about the minimum and maximum password length. The extracted requirements are finally stored in the PPML file format.

Outline

This thesis is organized as follows. After a short presentation of related work in Section 2, the necessary background information for this thesis is provided in Section 3. It comprises a brief description of the Password Policy Markup Language (PPML), a structured representation of password requirements, and the Unstructured Information Management Architecture (UIMA), a framework for information extraction. These sections are followed by the main contribution of this thesis:

Section 4 provides a high-level overview of the architecture of the PPC and briefly describes the application flow. The PPC consists of two separate modules, namely the Content Crawler (CC) and the Password Policy Extractor (PPE). The CC is responsible for searching the signup page of a service's website and is described in Section 5. Section 6 describes the PPE that processes the signup page in order to extract requirements and create a password policy.

Section 7 describes a conducted large-scale crawling of password policies. A list of one million websites were crawled and password policies for 72,125 domains were created. Section 8 introduces a password generator that can use the created password policies in order to generate random passwords that comply with a service's requirements.

Finally, Section 9 concludes the thesis and provides future work.

2 Related Work

Password generators can help users selecting secure random passwords. Password generators are an integral part of nearly all password managers and also exist as web (e.g. random.org [9]) and stand-alone applications (e.g. PWGen [10]). In general, they allow setting basic requirements such as the password length or the allowed characters. Furthermore, some password generators provide more advanced settings such as the password being pronounceable (e.g. Wesmid86) or that the generated password should not contain ambiguous characters (e.g. uppercase i and lowercase L). However, none of the current password generators allow the generation of passwords for a specific service. This means that users need to find out the requirements and configure the generator manually.

Shay et al. [11] present a formal language for password policies based on the generic authentication policy language AuthSL [12]. A simulation model that can be expressed based on the language can test the impact of a policy regarding its security before it is deployed in practice. However, the language cannot be used to express password requirements of real services because it does not provide crucial information such as the allowed character sets. Shay et al. [13] extended their simulation model by technical and human factors essential to the creation of password policies, but it still focuses on the simulation model to test the impact on security for existing services.

The service at passrequirements.com [14] lists the password requirements of web services in a list that can be searched by typing in a domain name. The requirements are provided as text that is copied from the service's website. Apart from the very limited list of currently only 64 services, the requirements are not published in a standardized format and not available through a standardized interface. To look up the requirements of a service, users need to visit a second website. Furthermore, because the service lacks an API, applications can neither retrieve nor process the password requirements automatically.

Egelman et al. [15] performed a laboratory experiment to see whether password meters influenced the user's behavior in their selection of weak or strong passwords. Users were forced to change their password in a real application not knowing that they were subject of a study. Egelman et al. conclude that password meters can lead users to select stronger passwords for "important" accounts. For low-risk accounts, users tend to reuse the same weak passwords. Password meters can therefore support the selection of stronger passwords, but users still reuse these passwords for a multitude of accounts.

Other services such as craigslist.com [16] try to overcome the problem of users selecting weak passwords that get reused for multiple services by removing the password selection completely. When registering for the service, users must only provide their email address and get a unique random generated password sent by email. This helps to enforce strong and unique passwords, but has further security implications. Because users tend to keep the email containing the password in their inbox, an attacker with access to the inbox could gain access to these accounts. In contrast to services where the passwords are not contained in such emails, the attacker would not need to use the reset password functionality. Therefore, the attacker has a higher chance to remain unnoticed by the user.

Single sign-on (SSO) mechanisms such as OpenID [17] and Facebook Connect [18] are designed to provide secure authentication among connected services without being prompted for different usernames and passwords. Urueña et al. [19] analyze the privacy of these SSO systems and state that both systems have privacy issues that make SSO mechanisms not a fully adequate replacement for multiple passwords. Furthermore, Wang et al. [20] discovered serious logic flaws that allow attackers to sign in as the victim user. They conclude that the overall security quality of SSO systems seems worrisome.

3 Background

The following chapter provides necessary background information for this thesis. First, the Password Policy Markup Language (PPML) is described in Section 3.1. PPML allows the specification of password requirements in a standardized format. Second, the Unstructured Information Management Architecture (UIMA) is described in Section 3.2. UIMA provides a framework for information extraction and is used as basis for the development of the PPC.

3.1 Password Policy Markup Language

The Password Policy Markup Language (PPML) is a XML-based data format used to specify password requirements for a service in a machine-readable format. One of the objectives of a so-called password policy[1] is to provide all information that is necessary in order to automatically generate secure and accepted passwords for a service. PPML focuses on the perspective of users and applications regarding the use of passwords. It does not provide information about a service's security measures for storing passwords and protecting user data.

In the following, details about the information that can be stored with the usage of a password policy are provided. Furthermore, a detailed description of the implementation of PPML is given. The PPML schema as used in this thesis can be found in the appendix Section B.

3.1.1 Functionality

A password policy can be divided into three parts (see Figure 3.1) which are described in the following. First, meta data contain information about the scope and currentness of a policy. Second, information about password requirements are stored in the policy such as the minimum and maximum length of the password. Third, PPML allows the description of routines used for password management. Routines can be used to automate management functionalities such as the changing of a user's password.

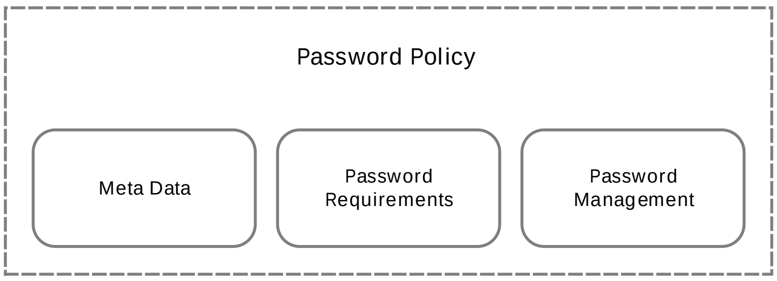


Figure 3.1.: Structure of a password policy.

[1] In this thesis, the terms *password policy* and *policy* are used interchangeably.

3.1.2 Meta Data

The meta data section contains additional information about the password policy.

- *Scope*: Defines the location for which the policy is applied. A policy can either be valid for a complete domain (e.g. http://example.com) or only a part of it (e.g. http://example.com/service1/). The possibility to define fine-grained scopes can be used to have password policies for different services residing on the same domain.
- *Name*: The name of the service the policy is applied to (e.g. Google). It is used to display a user-friendly identifier for the service to the user.
- *Version*: The version number of the policy. It allows applications to differentiate between multiple versions and to decide which policy to use in the case of multiple existing policies.
- *Timestamp*: The timestamp when the policy was created. It can be used to decide whether the using application should check for a new version.

3.1.3 Password Requirements

The password requirements section contains information that can be used in order to automatically generate passwords in accordance to a service's requirements. PPML allows the specification of the following requirements:

- *Minimum and maximum password length* (e.g. the password must have more than six characters)
- *Character sets* (e.g. the password can contain letters, numbers, and special characters)
- *Character restrictions*
  - *Minimum and maximum occurrences* (e.g. the password must contain at least one uppercase letter)
  - *Position restrictions* (e.g. the first character of the password must be alphanumeric)
  - *Consecutive characters* (e.g. password must not contain two identical consecutive characters)

There are very few services that have requirements which cannot be expressed using PPML. For instance, a service might use a blacklist to restrict the use of common passwords or passwords that contain parts of the username or email address. The password policies are intended to be used for the creation of random passwords and will therefore most likely not generate such invalid passwords. As described in Section 2 it is also well-known that blacklists are error-prone because they usually contain passwords for a single language only [21] and accept slightly changed common passwords such as *password!*.

3.1.4 Password Management

Besides information used for generating random passwords, a password policy can contain management routines that can be used in order to automate password-related operations. The management routines can describe three operations:

- *Login*
- *Password Change*
- *Password Reset*

A management routine contains a set of instructions that is sequentially executed. An exemplary login routine first loads a service's login page (e.g. https://example.com/login/). Afterwards, the username and password is filled in the corresponding input fields and the login form is submitted. Each instruction can include a set of assertions to ensure that the login was performed successfully. For instance, an assertion can check whether the website has set a certain cookie that allows the distinction between a successful and failed login.

3.1.5 Implementation

PPML is implemented as a XML Schema Definition (XSD) which allows the specification of password policies in the XML format. XML is a widespread format that allows the description and exchange of data on the web [22].

The XSD defines the structure of a password policy. Elements for the minimum and maximum password length can be represented as integer value or omitted if the service has no restrictions. Character sets are defined prior to the settings for requirements on minimum and maximum occurrences. A character set can either be defined by using a list of characters or by referencing one or more previously defined character sets. For each character set, a minimum and maximum amount of occurrences in the password can be defined (e.g. your password must contain at least one number). Restrictions on certain character positions in the password enable the setting of allowed characters depending on the character position (e.g. the first character must be alphanumerical). Each position restriction defines a set of positions to which the restriction is applied. This can be a single position as well as a comma separated list of multiple positions. Because the password policy may not specify an exact password length, negative positions define the character position starting from the end of the password. For instance, a character position of -1 refers to the last character of the generated password. Furthermore, the grouping of requirements regarding character sets allows the definition of rule sets. This allows even more complex password requirements that can be encountered on some services (e.g. use at least three of the following four rules).

The scope of a password policy is specified as an URL. It is also possible to refer to another URL in order to reuse an existing password policy. This allows the definition of a single password policy that can be applied for different scopes (e.g. example.com, example.org). Precisely, this means that the password policy for example.com can contain the complete description of password requirements. The password policy for example.org only specifies its scope and the redirect attribute linking to the existing policy. The reuse of existing password policies can therefore simplify maintenance because only one password policy has to be maintained.

The name and the version number can be any string and must not follow predefined conventions. The timestamp must be specified in the XSD datetime format [23] containing a combination of date and time as specified in ISO 8601 [24].

The XSD allows the definition of four different routine types that differ in their required technology. First, the HTTP routine type is able to send and receive plain HTTP requests (i.e. POST and GET). Second, the HTML routine type can additionally fill out forms that are received by the HTTP requests. This allows the submission of necessary hidden input fields where plain POST and GET requests are not sufficient. Third, the JavaScript routine type has enabled JavaScript support, e.g. for forms that are submitted using JavaScript. Finally, the Extended JavaScript type contains pure JavaScript code that is executed and therefore allows a broad interaction with the website. For management operations that require additional user input, such as CAPTCHAs or security questions, the routines allow the definition of placeholders that are requested from the user during the execution of a routine.

3.2 Unstructured Information Management Architecture

This section gives an introduction into the Unstructured Information Management Architecture (UIMA). After a short introduction into the term *unstructured information* and the difficulties associated with the extraction of information, a high-level architectural overview of UIMA is provided. UIMA is the underlying framework that has been used in the process of developing the PPC.

3.2.1 Unstructured Information

This section gives a brief introduction in the topic of unstructured information and the techniques that are currently used in order to process this kind of information. The term unstructured information (or *unstructured data*) refers to content that is not organized in a pre-defined data model. Unstructured information typically includes text or multimedia content such as emails, presentations, and webpages. In contrast to structured information, such content cannot be fitted into a relational database that can be used to directly access information. For example, a database might store the sender and recipient of an email in a database. Although the content of the email could also be stored, it cannot be used to access certain information contained in the text. This means that the database cannot be queried to answer questions regarding the content of the email (e.g. where is the meeting taking place?).

Further examples of unstructured information are:

- *Text files* (e.g. PDF files, webpages, or written documents such as books and letters)
- *Audio* (e.g. recordings for digital assistants such as Siri or Cortana)
- *Video* (e.g. news recordings)
- *Images* (e.g. scientific illustrations)
- *Presentations* (e.g. PowerPoint files)

The World Wide Web contains a seemingly unlimited amount of information, the majority of it represented as unstructured information. The active research in the field of Artificial Intelligence (AI) and

Natural Language Processing (NLP) focuses on the task of Information Extraction (IE) [25, 26]. Information Extraction is the task of automatic extraction of information from unstructured text such as types of events, entities, or relationships from textual data [27].

The PPC introduced in this thesis (cf. Section 4) uses IE techniques in order to extract password requirements from unstructured information. The unstructured information is the content that is presented to the user, which is present as natural language. Password requirements that are present as unstructured information are extracted using the UIMA framework which explained in the following section.

3.2.2 High-Level Architecture

In the following, the high-level architecture of UIMA is presented. After a short introduction in Unstructured Information Management (UIM) applications, the key components of an application using the UIMA framework are explained.

UIM applications are software systems that serve the purpose of analyzing large amounts of unstructured information to discover, organize, and extract relevant knowledge [28]. For knowledge extraction, UIM applications use a wide range of technologies, such as Natural Language Processing [29] (NLP), Information Retrieval [30] (IR), Machine Learning [31], and Automated Reasoning [32]. In particular, the unstructured data must be analyzed to interpret and detect information of interest that are not explicitly tagged for automatic processing, such as named entities [33], organizations, or locations. For example, an UIM application could detect persons involved in a text book. More sophisticated applications might find opinions in user product reviews to support business intelligence.

UIMA is a software framework that supports the development of such UIM applications. The architecture of UIMA provides components and data representations essential for UIM applications. Figure 3.2 shows the architectural high-level view of the document-level analysis using the UIMA framework.

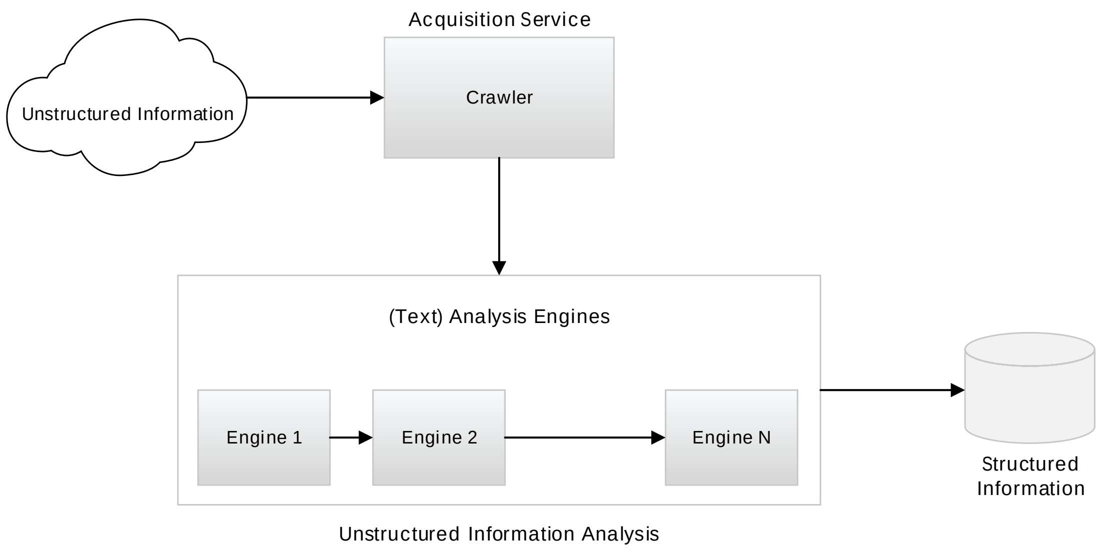


Figure 3.2.: High-level architectural overview of UIMA.

An acquisition service produces a document collection that will be analyzed by the application. For example, this may be a web crawler that has crawled a set of websites of interest. The variety of applications that may provide collections to UIMA is not limited by the architecture. So-called *Collection Readers* must provide the interface to access a collection's elements.

The documents are then accessed by the *Analysis Engines* (AEs) or respectively for text by the specialized *Text Analysis Engines* (TAEs). A TAE is a recursive analysis structure that can itself contain multiple TAEs. Each TAE represents a certain analysis step that focuses on the discovery of specific concepts such as the recognition of named entities. The analysis that is produced as well as the original document are stored in the *Common Analysis Structure* (CAS). Information that is added to the CAS is represented as *annotations*. An annotation is a metadata structure that associates information with a span of text in the document [28]. The CAS is more and more enriched with information as it is passed along the stages of analysis.

The architecture of UIMA allows developers to focus on the development of algorithms that will be encapsulated in an analysis engine. The analysis engine can expect the presence of certain information in the CAS (referred as *CAS in*) and produce a certain output (referred as *CAS out*). This supports the reuse of components across multiple applications.

4 Password Policy Crawler

In this section, a solution to generate password policies automatically is presented. The Password Policy Crawler (PPC) is a software application that extracts the password requirements from a service's website and creates the corresponding password policy automatically. First, a brief overview of the functioning of the PPC is provided. Second, a high-level overview of the architecture and components of the PPC is given. Third, details about the evaluation set used for evaluating the key modules of the PPC are presented. The key modules are later described in detail in Section 5 and 6.

The PPC creates a password policy for a given service in the following four steps:

1. Given a service's domain (e.g. example.com), the PPC finds the signup page and stores its HTML source code.

2. The source code of the signup page is processed and information about the minimum and maximum length of the password is extracted from the password input field.

3. Requirements that do not refer to the password (e.g. username requirements) are removed. The remaining text is parsed and password requirements are extracted from texts such as "Use six or more characters".

4. Finally, a password policy is created specifying the found password requirements.

The architecture of the PPC is illustrated in Figure 4.1. The creation of a password policy for a given domain is divided in two key modules. The Content Crawler (cf. Section 5) is responsible for finding the signup page of the given domain. The HTML source code of the signup page is stored in a local database that is accessed by the Password Policy Extractor (cf. Section 6). In the extractor, two approaches are used for extracting password requirements. First, the password requirements that are present as natural language (e.g. "Use at least one lowercase character") are extracted. Second, attributes present in the HTML password input field can be used to extract information about the minimum and maximum password length.

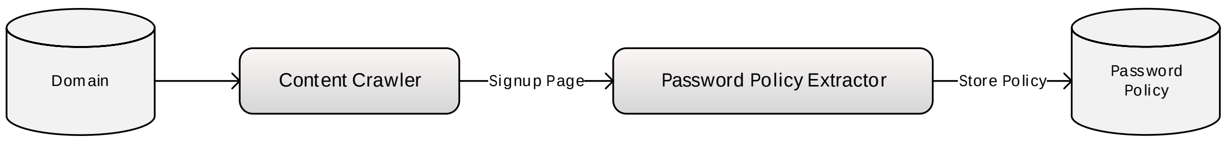


Figure 4.1.: High-level overview of the Password Policy Crawler.

Evaluation Set

The following section describes an evaluation set that was used to evaluate the PPC. The evaluations performed on the two key modules of the PPC in Sections 5.2 and 6.2 use a list of domains with English-speaking websites which all have a publicly available signup page. The set of domains used for evaluation was derived from the Alexa Top 500 US list [34] in a multi-step process. The list represents the top most visited websites from the United States.

First, all non-English websites were filtered out. Second, websites containing illegal and/or adult content were identified and removed from the list. Third, the remaining websites were analyzed manually whether they contain a signup page. Besides websites that do not offer any possibility for users to create an account, there are websites that offer registration only with an existing offline account (e.g. media companies [35] or online banking) as well as websites that require the input of payment information prior to registering. Table 4.1 shows the amount of websites removed per filtering process. In total, this results in 200 websites respectively domains that were used for the evaluation. The complete list of domains can be found in the appendix Section D.

Filter	Removed Websites	Remaining Websites
Non-English	7	493
Adult/Illegal content	13	480
No public signup	280	**200**

Table 4.1.: Filtering for domains with public signup pages among the Alexa Top 500 US list.

5 Content Crawler

In the following, the Content Crawler (CC) is described, which finds the signup page of a service. After a short description about the functionality of the crawler, detailed information about the implementation is provided. Furthermore, an evaluation of the crawler is presented. It will be shown that the crawler is able to achieve an overall good precision of over 94% in finding the signup page for a given domain. Additionally, findings during the evaluation are outlined and discussed.

As input, the CC receives the domain of a service, e.g. example.com. It queries a search engine to find possible signup pages. It accesses the first three results retrieved by the search engine and checks, whether one of the pages is a signup page. If no signup page is found on all candidates, a further analysis of links contained on the visited pages and matching certain patterns is performed. Analogous to pages retrieved from the search engine, the links are visited and a search for signup pages is performed. Found signup pages are stored in a local database for later access and further processing by the Password Policy Extractor (cf. Section 6).

5.1 Implementation

For a given domain, the CC first queries a search engine to retrieve URLs to a possible signup page. These URLs will be visited and searched by the Signup Page Detection component which analyzes the HTML structure in order to decide whether a page is a signup page. If no signup page is detected, the Link Extraction component extracts anchor links that match certain keywords, such as “Join” and “Sign up”. Only if no signup page can be found using the URLs retrieved from the search engine, the extracted links will be checked. As a last step, the Content Extraction component visits a found signup page using a common browser in order to extract a processed HTML Document Object Model (DOM) structure. The structure of the crawling module is illustrated in Figure 5.1.

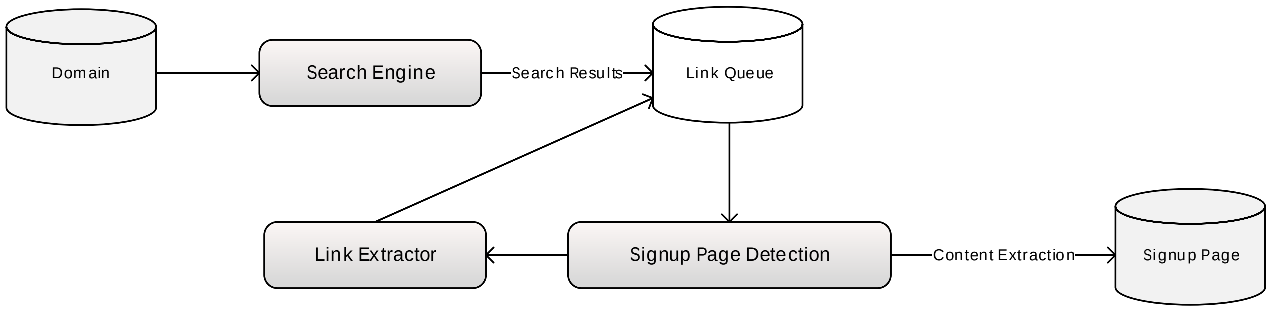
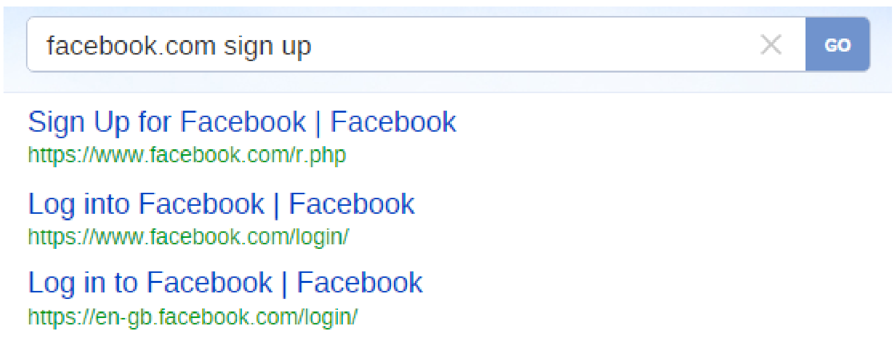


Figure 5.1.: Architectural overview of the Content Crawler.

5.1.1 Search Engine

The Search Engine component uses a search engine web service in order to find URLs to possible signup pages. To retrieve relevant results, a simple query containing the domain name of the service as well as the phrase “sign up”. For example, for the domain example.com the component would build the query “example.com sign up”.

This component uses the StartPage web service [36] as search engine which retrieves its results directly by the Google search engine. In contrast to Google, StartPage has a more liberal restriction regarding the maximum allowed queries per day. However, the use of other search engines would likewise be possible. Figure 5.2 shows a typical query and its results using the StartPage web service.


facebook.com sign up

GO

Sign Up for Facebook | Facebook
https://www.facebook.com/r.php

Log into Facebook | Facebook
https://www.facebook.com/login/

Log in to Facebook | Facebook
https://en-gb.facebook.com/login/

Figure 5.2.: Sample StartPage search as used by the CC [37].

5.1.2 Signup Page Detection

In this section, the solution how to detect a signup page is presented. The Signup Page Detection component decides, whether a URL points to a signup page or not. In order to do so, the URL is being accessed and scanned for the HTML form elements (`<form>`) [38]. The component checks if the page contains a form that can be used to create an account (i.e. a signup form). In general, a form element defines where and how the user input is submitted that is provided by filling contained input elements. However, not every form represents a signup form. Forms may serve different purposes such as:

- Login
- Signup
- Newsletter subscription
- User settings

Therefore, the form is processed and analyzed to fulfill certain requirements to be considered as signup form. In a first step, all `input` and `select` elements are extracted and analyzed. The set of input fields is then filtered by their type to remove elements of non-text type such as checkboxes, images or buttons. The remaining set of form elements must finally fulfill the following requirements to be considered as a signup form:

- At least one input of type password is present.
- The set is at least of size 3.

The CC expects the set to have at least three input fields to distinguish between a login and signup form. While a login form usually only asks for a username and password, a signup form requires the input of more information such as the user's email address, its name, and/or birthdate.

For accessing URLs, this component uses the Selenium framework [39] in order to send a HTTP request to the corresponding web server and analyze the parsed result for forms of interest. Selenium is a testing framework that provides functionality for automating browser interaction. It uses browsers such as Firefox or Google Chrome in order to interact natively with websites. In contrast to using plain HTTP requests and directly analyzing the DOM structure from the server response, it is possible to retrieve the DOM structure after it has been processed by the browser. This is especially beneficial for websites that use JavaScript and AJAX [40] because these technologies might significantly modify the source code due to delayed content presentation or asynchronous downloading of additional resources. As a result, the DOM structure that is received through the HTTP server response might not contain all the information that would finally be presented to the user.

5.1.3 Link Extraction

In some cases, the retrieved search results from the search engine are not directly referring to a signup page. To address this issue, the Link Extraction component parses all links present on the pages that were retrieved from the search engine. The relative anchor links and link texts are then matched against a small dictionary to find links of interest. Links of interest contain keywords such as `Sign Up`, `Create Account` or `Join Now`. The found links are then added to the queue to be processed.

Overall, the pages that are processed build a prioritized order, where as soon as the signup page has been found, all remaining pages in the queue will be discarded. Before any extracted link is processed, all search results retrieved from the search engine are processed.

There are several reasons why the search results may be inaccurate for the intended goal. One of the reasons is the HTML robots meta tag [41]. The robots meta tag allows webmasters to control the behavior of search engines when crawling a website. Setting the content attribute tag to "noindex" tells the visiting search engine not to index a certain URL and therefore not to present them in the search results.

There are services that have this tag set on their website's signup page. An example of a service with a very large user basis is Wikipedia. Figure 5.3 shows the relevant section of the HTML structure that is present in the signup page on wikipedia.org.

```
<head>
        <!-- ... -->
        <meta name="generator" content="MediaWiki 1.27.0-wmf.2" />
        <meta name="robots" content="noindex,nofollow" />
        <!-- ... -->
</head>
```

Figure 5.3.: Robots meta tag as present on wikipedia.org [42].

Analogous to the robots meta tag, there exists the robots.txt [43] that webmasters can provide from a well-known location [44] of their web server. Both approaches are considered by all major commercial search engines [45] and can therefore lead to inaccurate search results.

Another reason may be a bad overall search engine optimization on the used query and the keywords found on the signup page of a website. A service's website might contain multiple pages that all contain the queried keywords (sign up), but not all of them are actually signup pages. An example for this case is the iStockPhoto service. In addition to the signup page, istockphoto.com has a large amount of pages that are showing commercial pictures that are tagged with the keywords `sign up` (see Figure 5.4). The fact that users commonly use keywords to find photos on iStockPhoto leads to a high ranking of one of these keyword pages in the search results.

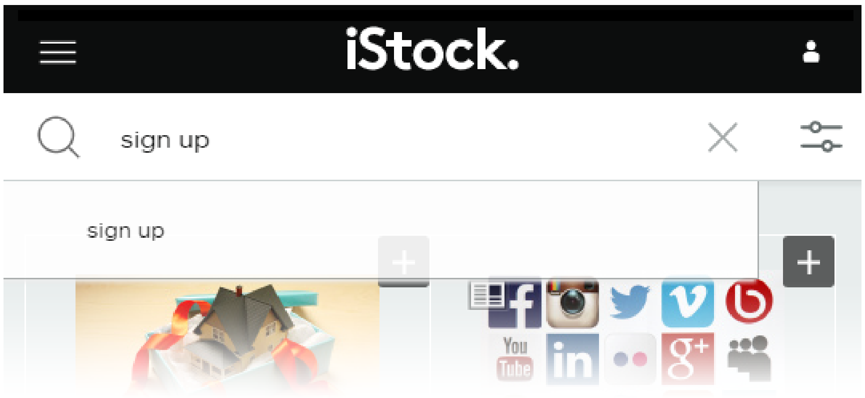


Figure 5.4.: iStockPhoto images page found by the search engine [46].

Using the Link Extraction component, the processing order of links for iStockPhoto then looks as follows:

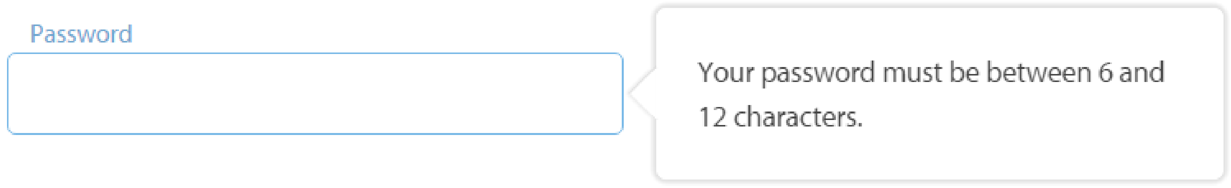
- `http://www.istockphoto.com/sell-stock-photos.php`
- `http://www.istockphoto.com/photos/sign+up`
- `http://www.istockphoto.com/`
- `https://secure.istockphoto.com/join/aHR0[...]`

The last link represents an extracted link from the first search result. After content is passed to the Signup Page Detection as described in Section 5.1.2, it is recognized as signup page and stored in the local database to be parsed for password requirements. All remaining links in the queue will be discarded and the processing is stopped.

5.1.4 Content Extraction

The final operation of the CC is the storage of found information. After the signup page has been loaded by the browser, the password input field is focused and the processed HTML source code is stored.

After a signup page has been loaded, there are many other resources besides the retrieved HTML that could be stored for later requirement extraction. Typically, a web page does not only consist of the received HTML code from the request sent to the server. Inside the HTML code, other resources like images, JavaScript resources, and stylesheets are referenced. Disregarding this information might result in a loss of information about the password requirements. Figure 5.5 shows a screenshot of the password input field of the signup form from walmart.com. It can be seen that the password requirements are shown as a tooltip. This tooltip appears as soon as the password input field is focused by the user.



Your password must be between 6 and 12 characters.

Figure 5.5.: Password requirements as outlined on walmart.com [47].

Analyzing the received HTML source code from the server, these requirements are not represented in the content. Instead, it is dynamically added via a JavaScript as seen in Figure 5.6. The example shows a case where phrases are stored in a JavaScript array containing language variables. However, the additional parsing of JavaScript would not be expedient for two reasons:

First, many websites use JavaScript where the language variables for form validation contain placeholder variables that are dynamically filled. This would require complex analysis of the code in order to find out what values will be filled into the language variables.

Second, it is not always clear whether a requirement is affiliated to a password or other required inputs such as the username or email address. A string contained in JavaScript with the contents “Six or more characters” cannot easily be associated to the object it is referencing. Without further knowledge, the requirement may also be associated to the username that must be chosen.

```
return {
        "common.sign-up.password-text": "Password",
        "common.sign-up.password-flyout": "Your password must be between 6 and 12 characters.",
        "common.sign-up.confirm-password-text": "Confirm password",
}
```

Figure 5.6.: Tooltip text inserted from JavaScript array [48].

Therefore, the password field is set to be focused by Selenium in order to have these requirements included in the resulting source code. This eliminates the need of storing additional resources such as external JavaScript files.

5.1.5 Limitations

There are several website structures that currently cannot be processed successfully by the CC which are described in the following.

Country Restrictions

First, there exist services that can only be accessed from certain countries, also known as geo-blocking [49]. Service providers make use of this features for different reasons like licensing constraints, legal issues (e.g. online casinos) or to block access to online shops that have complementary services in a user's country. For finding signup pages, the CC was completely operated from Germany with the aim to crawl English websites. This led to some websites being unable to be crawled due to country restrictions, for instance pandora.com:

> "*We are deeply, deeply sorry to say that due to licensing constraints, we can no longer allow access to Pandora for listeners located outside of the U.S., Australia and New Zealand. We will continue to work diligently to realize the vision of a truly global Pandora, but for the time being we are required to restrict its use. We are very sad to have to do this, but there is no other alternative.*
>
> *We believe that you are in Germany (your IP address appears to be 87.167.3.153). If you believe we have made a mistake, we apologize and ask that you please email us.*" [50]

Nonetheless, operating the crawler from another country or making use of a VPN service can only be a partial solution. Because also non-US sites make use of geo-blocking, the crawler would have to know from where a site can be accessed *prior* to visiting the site.

JavaScript overlays
Second, the crawler does not interactively process websites using JavaScript and AJAX. As modern technologies such as HTML5 and AJAX get more and more widespread on the internet, there exist websites that cannot be used without JavaScript support. Furthermore, there are also registration forms that are only being loaded asynchronously as modal window shown on the current page when clicking on the sign up link. Because the crawler treats every extracted link as a separate new page load, these websites can in some cases not be recognized or accessed.

Endlessly loading websites
Third, there exist websites with content that cannot be extracted with the use of the Content Extraction component and Selenium. These websites usually have a very high advertisement load. If not properly implemented, the loading of advertisements prevents a site from finishing loading. This will prevent JavaScript from triggering events such as showing tooltips when a password field is focused. Moreover, internal JavaScript will also fail loading asynchronous content such as password requirements. Tests showed that the majority of these sites are sites with adult content. Solutions such as using adblockers can only provide little assistance because they can also cause sites to not load properly. However, as it will be shown in the evaluation (see Section 5.2), these websites represent an overall minority.

Retrieval of password requirements
Another aspect is the retrieval of password requirements. The crawler focuses the password input before storing the source code of the web page. However, there are websites where this procedure is insufficient for retrieving all requirements. The most common limitation here are websites that do not show any password requirement until a form has been submitted. The user choosing the password does not know whether it is accepted until filling out and submitting the complete registration form.

Availability of signup pages

It is evident that the CC is also only able to crawl publicly available signup pages. There are different services that do not provide such signup pages, most of them are in the banking sector. These sites allow creating online accounts only with an existing offline account and can therefore not be crawled.

5.2 Evaluation

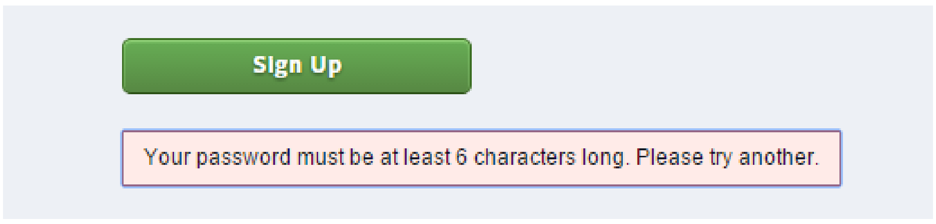
To be able to make a statement about the quality of the CC, the components were evaluated in a multi-stage process. This evaluation is described in the following sections. First, Section 5.2.1 discusses findings that were made during the evaluation. Second, Section 5.2.2 provides a detailed evaluation of the search results retrieved from StartPage. Third, Section 5.2.3 shows the improvement that is achieved by adding the link extraction. Furthermore, Section 5.2.4 discusses the final results and Section 5.2.5 concludes this chapter.

For all evaluations, the domains from the evaluation set as introduced in Section 4 were used. The set contains a list of 200 domains derived from the Alexa Top 500 US sites. All 200 domains are English and have a publicly available signup page. As stated in Section 1, the overall goal is to be able to generate secure passwords for a given service. Services without any signup functionality, for which the CC falsely found a signup page, should not be considered as an error because policies that would be created for these services would never be requested. Therefore, these erroneous policies would have no impact on the overall operation of the PPC.

5.2.1 Findings

This section provides findings that were made during the development and evaluation of the CC. One of the first questions that had to be answered with regard to developing such a software is where to look for the password requirements that will be mapped to a password policy. In general, it can be assumed that the requirements for a password can be found on the signup page of a given web service. This is due to the fact that a user, registering for the service, has to be informed about the requirements that the chosen password has to fulfill in order to be accepted.

A manual evaluation of the Alexa Top 500 US sites showed that Facebook was the only service that provided detailed password requirements only on a dedicated page. Facebook uses an algorithm which identifies weak passwords that will be, eventually, rejected for use. When registering for the service, the only password requirement that is shown is the minimum password length in case of submitting the form with an invalid password as seen in Figure 5.7.



Your password must be at least 6 characters long. Please try another.

Figure 5.7.: Error after submitting an invalid password to Facebook [51].

Looking through the Help Center, a dedicated page for password requirements can be found with a more detailed listing of password requirements:

> "***What is the minimum password strength and how can I make my password strong?***
>
> *When you create a new password, make sure that it's at least 6 characters long. Try to use a complex combination of numbers, letters and punctuation marks.*
>
> *If you see a message letting you know the password you entered isn't strong enough, try mixing together uppercase and lowercase letters or making the password longer.*" [52]

To reduce the complexity of the PPC, only the signup pages are analyzed for password requirements. This approach is sufficient in order to provide correct password policies for the majority of evaluated services including Facebook and Google. However, the CC could easily be adapted to search for different topics than the signup page.

5.2.2 Search Results

The CC only retrieves the first three search results of a query that is being sent to StartPage. This is as well for speeding up the process of finding signup pages as due to the assumption that the used search engine returns the most relevant pages for the query in the top ranks.

Therefore, this evaluation serves two purposes. On the one hand, it is necessary to prove that the used search engine is able to deliver accurate results and can be effectively used to retrieve URLs to signup pages. On the other hand, it must be shown that the limitation to three results per query does not significantly lower the quality of search results. This limitation speeds up the process of finding signup pages, which is especially important for large-scale crawling. If a large amount of signup pages is located after the third search result, the crawler would need to be reconfigured to retrieve more results. Furthermore, if signup pages cannot be found for a large amount of pages through a search engine at all, this approach would have to be reconsidered.

In this evaluation, the query as described in Section 5.1.1 was submitted to the search engine and the first thirty results obtained were reviewed manually. Table 5.1 shows the distribution of the average rank of the correct URL to the signup page for the evaluated websites. For the evaluated websites, 136 signup pages, which equals a percentage of 68, were retrieved with the first search result. The second search result provided the URL to a signup page for 22 of the remaining 64 websites, which equals 11%. A small amount of 4 signup pages (2%) were found through the third search result. This adds up to a total of 81% of signup pages that could be found using only the first three search results. Furthermore, it can be seen that very few signup pages can be found using more than the first three search results. The use of up to eight search results would lead to a theoretical improvement of 1%, while an improvement of 1.5% may be possible by the use of all thirty evaluated search results. It must be noted that the statistic includes only the best found rank for each website, even though correct URLs may be returned for multiple ranks.

Search Rank	**Websites**	**Percentage**	**Percentage aggregated**
Rank 1	136	68%	68%
Rank 2	22	11%	79%
Rank 3	4	2%	81%
Rank 4-8	2	1%	82%
Rank 9+	1	0,5%	82,5%
Not found by the search engine	35	17.5%	100.0%
Total Signup Pages	*200*	*100%*	

Table 5.1.: Evaluation of the average first signup page position.

These results strengthen the thesis that there exists only a small tradeoff between returning only the first three results and the speedup of the overall process. It also shows that the majority of sites can be found with querying a search engine at all. The use of up to thirty search results would only lead to a theoretical improvement of 1.5%, while this would on the other hand have a large impact on the operating speed of the crawler. As stated in Section 5.1.3, reasons for search engines for not finding a signup page can be the `noindex` meta tag or a bad overall search engine optimization.

5.2.3 Link Extraction

For evaluation of the link extraction component, the same 200 domains as for the search engine evaluation were used. It is measured, in how far the extraction of matched links can be used to increase the accuracy of finding a signup page. Table 5.2 shows the improvement from the StartPage search in combination with link extraction.

Search Rank	**Websites**	**Percentage**	**Percentage aggregated**	**Improvement**
Rank 1	155	77.5%	77.5%	11.5%
Rank 2	23	11.5%	89%	0.5%
Rank 3	4	2%	91%	
Rank 4-8	2	1%	92%	
Rank 9+	1	0.5%	92.5%	
Total Signup Pages	*200*	*100%*		

Table 5.2.: Evaluation of the average first signup page position in combination with link extraction.

It is noticeable that the vast majority of found signup pages through link extraction is found using the first search result. For search rank 1, the improvement sums up to 155 (77.5%) found signup pages. Furthermore, 23 (11.5%) pages could be found through the second search result. For all other search results, no further improvement could be achieved.

Overall, there are only 18 (9%) of the evaluated services left for which no signup page could be found using the presented method. There are different reasons why the combination of the two discussed components are not sufficient to find a signup page (see Table 5.3).

Reason	Websites	Percentage
Country Redirect	2	1%
Missing Keywords	3	1.5%
AJAX/JavaScript	9	4.5%
Other	2	1%

Table 5.3.: Reasons that signup pages are not found by the Link Extractor.

The country redirect, as discussed in Section 5.1.5 cannot easily be solved. Missing keywords mostly occur on websites that use a graphical button for signup links or a very special captioning of these links (e.g. "Create an Adobe ID" [53]). The main reason however is the missing interaction with the websites. Around 5% of websites use sign forms that are asynchronously loaded and shown as a modal window to the user after a click on a signup link, for instance on cnbc.com (see Figure 5.8).

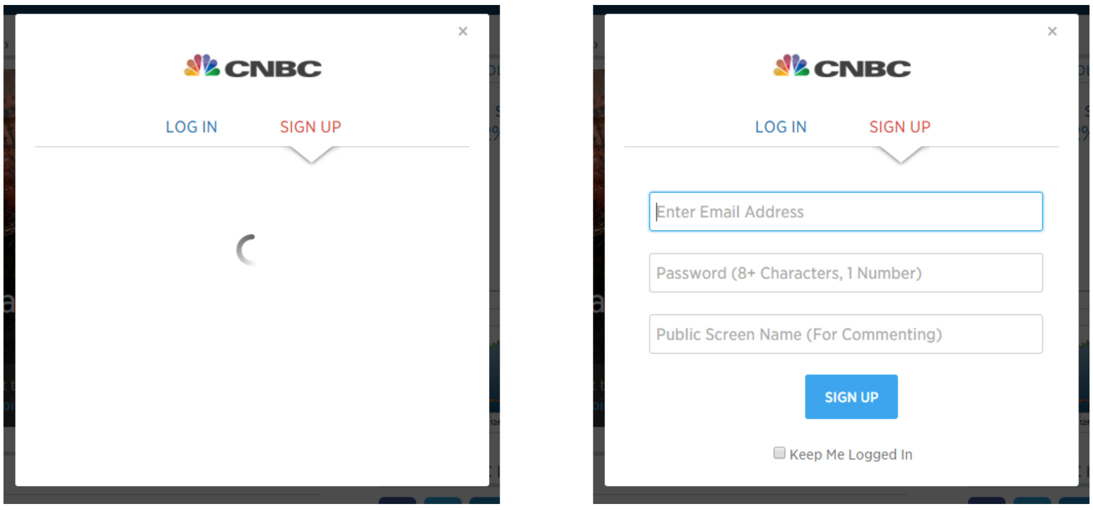


Figure 5.8.: Asynchronous loading of the signup form on cnbc.com [54].

As described in Section 5.1.5, the CC does not interact with the website in a way that these websites can be crawled. This topic will be discussed in the future work Section 9. Other reasons might be very unique constructions of a signup functionality, such as a preceding required form submission [55] to be redirected to the corresponding signup page.

5.2.4 Final Results

Being able to effectively retrieve the URL of a signup page is the second last step in detecting a signup page. Recognizing a page as such is evaluated in this section. This evaluation combines the results presented in the previously discussed sections and presents important key figures used for measuring the overall quality of the CC.

There are several characteristics that can be measured. In general, the CC can be considered as a classifier. Given a domain, it must decide whether there exist a signup page (*true*) on its website or not (*false*). If it decides that a signup page exists, the page must be returned. In this classification process, there are four possible outcomes. For a website that contains a signup page, the crawler can either return the correct page (in the following referred as *true positive* or *tp*) or a wrong/no page (*false negative* or *fn*). Respectively for websites that contain no signup page, the crawler can either return that there exists no page (*true negative* or *tn*) or a wrong page (*false positive* or *fp*). In the performed evaluation, no true negatives can occur because the evaluation was performed on the evaluation set (see Section 5.2) which contains only domains with an available signup page.

Out of these measures, important key figures can be derived. The precision p defines the percentage of documents retrieved that are actually relevant [56, 57]. It is defined as follows:

$$p = \frac{|\{relevant\ documents\}| \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|} = \frac{tp}{tp + fp}$$

Recall r [56] is the fraction of relevant documents among all retrieved documents and defined as:

$$r = \frac{|\{relevant\ documents\}| \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|} = \frac{tp}{tp + fn}$$

In case of the CC, precision measures the percentage of true signup pages among all pages that were classified as those. Recall on the other hand defines the percentage of signup pages retrieved among all existent singup pages. Running the crawler on the 200 domains used in the previous evaluations, the test outcomes as stated in Table 5.4 were observed.

		Condition	
		True	False
Test Outcome	True	149	9
	False	42	-

Table 5.4.: Classification results of the CC.

From these results, the precision and recall can be derived:

$$p = \frac{tp}{tp + fp} \approx 0.9430$$

$$r = \frac{tp}{tp + fn} = 0.745$$

The high precision of about 94% indicates that if the CC classifies a web page as a signup page, it is most likely a true signup page. This means that the majority of output pages are real signup pages. On the other hand, the in comparison lower recall of 74.5% means that the CC is not able to find signup pages for all given domains. For the usage of the CC, it can be stated that the precision is the more important factor, because it is necessary to trust the module that the given documents are correct. Incorrectly returned documents will otherwise lead to wrong password policies, whereas a low recall only leads to missing policies. In contrast to incorrect policies, missing policies can be easily recognized as such.

5.2.5 Summary and Conclusion

The evaluation has shown that the CC is able to provide accurate results for a large fraction of given domains. The accuracy of today's search engines allows a rather easy process of finding signup pages with a good overall precision. The precision indicates that the majority of extracted pages are real signup pages that can be used for creating password policies.

The application area of the CC is a very large amount of domains. Looking only at the currently registered domains within the com TLD, they represent a volume of over 100 million domains [58]. Given this large amount of domains, the importance of the recall fades into the background. The greatest challenge will be to set up an infrastructure that is large enough to visit a significant amount of these domains to crawl for password policies.

Nonetheless, solutions must be provided to overcome the gap of the recall value. Section 9 will discuss future work including improvements that can be implemented into the CC in order to increase the recall value. A community-based approach for creating missing password policies and correcting those that were not fully correctly crawled will be presented.

6 Password Policy Extractor

In this chapter, the Password Policy Extractor (PPE) is described in detail. The PPE extracts password requirements from a given document and creates a corresponding password policy. After a brief description on the functioning and the problem statement, implementation details are provided in Section 6.1. The subsequent evaluation in Section 6.2 will show that this extractor is able to provide correct password policies for 91,5% of the evaluated domains.

After signup pages have been found and extracted by the CC (cf. Section 5), the requirements on a password must be extracted out of the stored HTML document. The PPE uses Natural Language Processing (NLP) techniques in order to make such information readable for machines. Additionally, information about the allowed password length is extracted from the password input field. The information that is present in HTML documents is in general not organized in a pre-defined manner because it is not intended to be read by applications. Instead, this information is solely for the purpose of user-presentation. Such information that is not per se machine-readable is usually referred as unstructured information (cf. Section 3.2.1). In order to extract knowledge from these documents, approaches like regular expressions are usually not expedient because the structure of possible sentences containing the information must be known prior to parsing the content. To make unstructured information machine-readable, other approaches such as NLP must be used. This approach allows the extraction of information from sentences with structures that have not been processed before.

6.1 Implementation

The PPE is based on the UIMA framework (cf. Section 3.2) to process information, extract meaning and create structured data [59]. The pipeline layout used in UIMA is used to split the process of requirement extraction into multiple components, as illustrated in Figure 6.1. In UIMA, the components use a Common Analysis Structure (CAS, cf. Section 3.2), where extracted information is stored and accessible by proceeding components in the pipeline.

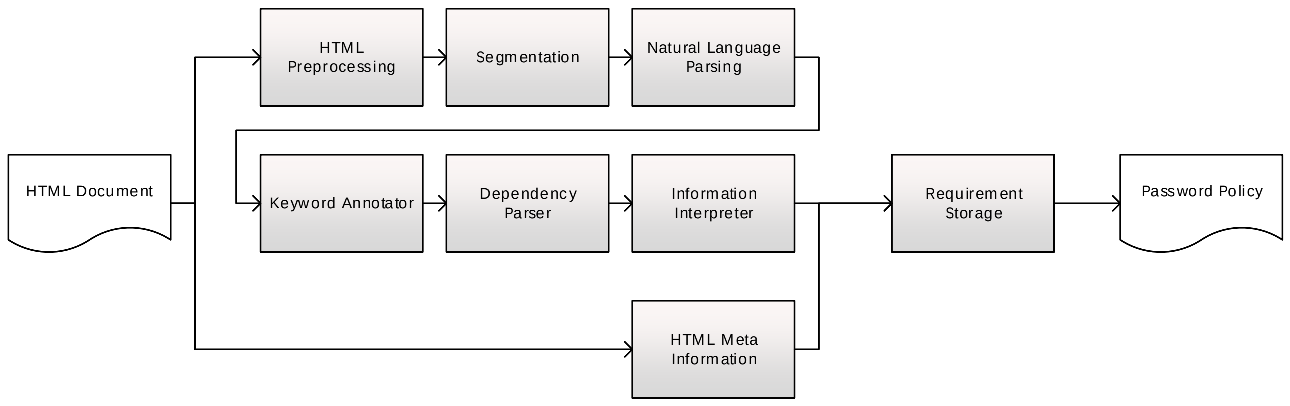


Figure 6.1.: Architectural overview of the Password Policy Extractor.

Beginning with a HTML document, the components process the document and pass gathered information along the pipeline. In a first step, the HTML source code is processed and relevant text is being extracted. This text is then analyzed and splitted into words as well as sentences. The Natural Language Parsing component analyses the sentences and is able to put the words of a sentence into a hierarchical tree that represents dependencies between words such as *subject* and *indirect object*. The subsequent components are able to use this information to find sentences that relate to password requirements. Out of these sentences the specific password requirements are extracted in order to finally store them in a XML representation using the PPML schema. Additionally, the structure of the HTML source code is used to extract information about the minimum and maximum password length using the password input field. In the following, these components are described in detail.

6.1.1 HTML Preprocessing

The HTML Preprocessor is the first component of the PPE module. It is operating on the HTML source code and prepares content for all other processing steps. Standard UIMA Text Analysis Engines (TAEs) generally require plain textual input [28]. However, the input that is provided by the CC is raw HTML and therefore needs to be transformed to a text-only representation that contains no HTML markup. In general, such a transformation is performed by stripping out all present HTML tags and returning the remaining text.

To improve information extraction results, the preprocessor uses a more complex algorithm in order to decide which information should be transferred to the text representation of the document. Overall, the HTML preprocessor serves three purposes that will be explained in the following:

- Filtering of unnecessary content.
- Filtering of text blocks containing requirements not related to the password.
- Providing sentence delimiters.

Because Natural Language Processing is a very computationally expensive task, the filtering of unnecessary content can significantly speed up the process of information extraction. For filtering, the HTML structure and element type system is being exploited. The nature of the information of interest, namely password requirements, allows the exclusion of certain HTML elements from the overall analysis. For instance, it is improbable for password requirements to be found inside a `select` HTML element that represents a control for selecting amongst a set of available options [60]. Altogether, the following elements are completely being discarded including the contained content:

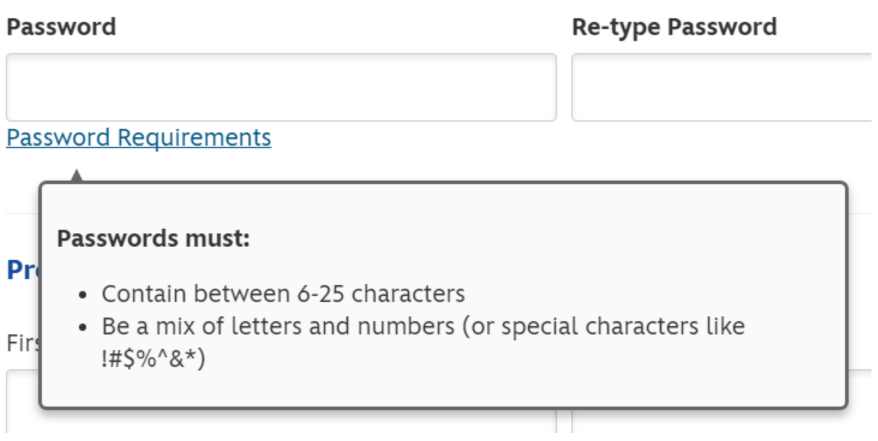
- head
- select
- script
- style
- iframe
- embed

Since the use of HTML5 gets more widespread, new elements and attributes are increasingly being used. One of these attributes is the `placeholder` attribute [61] that can be used within `input` elements. This attribute is shown as long as the input is not focused and is often used to provide information about the content that has to be filled in. Therefore, the placeholders for input elements of the `password` type may also be of interest and are also extracted. Additionally to the placeholder attribute, all `data-*` attributes [62] that contain the keyword *password* or an abbreviation of it will be extracted. Service providers use the `data-*` attributes to show conditional error messages that may also contain information about password requirements (see Figure 6.2).

```
<div>
        <p>Create Password</p>
        <input
                data-val-length="The field Password must be a string with a minimum length of 5 and a maximum length of 24."
                type="password" />
</div>
```

Figure 6.2.: Password requirements contained in the data attribute on ancestry.com [63].

A crucial part of the preprocessor builds the filtering of text blocks that most likely contain no password-related requirements. The HTML Preprocessor is therefore also the first component that decides whether text is referencing a password or any other subject. When parsing natural language, the extractor module is not tracking the subject a specific sentence is referencing, as long as it is not mentioned in that exact sentence. For example, the requirements of a password might be provided in a tooltip, where the keyword *password* is only mentioned once in the headline, as illustrated in Figure 6.3.



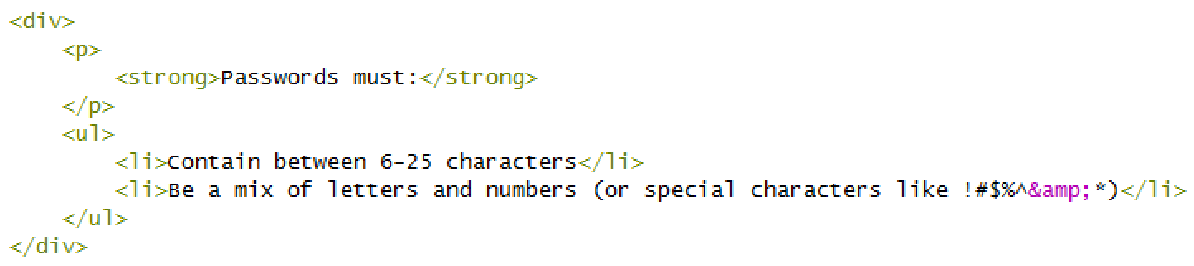
Passwords must:

- Contain between 6-25 characters
- Be a mix of letters and numbers (or special characters like !#$%^&*)

Figure 6.3.: Password requirements used on go.com [64].

However, such kind of tooltips might also be provided for requirements on other information like the username that can be chosen. To prevent the following components to extract requirements that are not referencing the password, they need to be filtered out. In order to do so, the hierarchical ordering of elements in the HTML structure can be used. Looking at the HTML source code of the provided example (see Figure 6.4), it is noticeable that the title containing the password keywords and the requirements share the same hierarchical parent.

The Preprocessor uses this fact and utilizes two small sets of keywords for filtering. The first set contains keywords that indicate that all elements in the hierarchical lower level should be filtered (*exclusion set*).



```
<div>
    <p>
        <strong>Passwords must:</strong>
    </p>
    <ul>
        <li>Contain between 6-25 characters</li>
        <li>Be a mix of letters and numbers (or special characters like !#$%^&amp;*)</li>
    </ul>
</div>
```

Figure 6.4.: Password requirements HTML markup on register.go.com.

The second set contains keywords that indicate that the hierarchical structure should not be filtered (*inclusion set*). In order to filter a given structure, it must contain keywords of the exclusion set, but none of the inclusion set. This results in a recursive structure starting from the root node of the DOM recursively descending until the conditions are met or a leaf node has been reached.

The last preprocessing step serves to improve the overall textual structure of the document. More precisely, this means the addition of sentence delimiters that are initially missing in the original text. Because further processing in later components of the pipeline require the detection of sentences, this approach can help achieving significantly better performance in these processing steps.

To decide where adding punctuation is needed, a distinction between different types of HTML elements is made. It can be reasonably assumed that a sentence is not spanning across multiple block level elements [65]. Additionally, punctuation is added after HTML labels and the `option` element. Examples of block elements are:

- div
- p
- form

Overall, these presented methods for preprocessing contribute significantly to the performance of components presented in the following.

6.1.2 Text Segmentation

Text segmentation is a common preprocessing task in NLP applications. It is the first step in parsing sentences of natural language and describes the process of dividing text into meaningful chunks of data. For the PPE, this process is divided into two steps.

The first segmentation step is the word segmentation. Word segmentation is the problem of finding word boundaries in a given text. For the English language, this is a rather straight-forward process because there exist orthographic spaces between words [66]. The output of the word segmentation are annotations that contain information about where a word begins and where it ends.

Because further processing steps also require the input to be divided into sentences, the second segmentation step is the splitting of texts into sentences, known as Sentence Boundary Disambiguation (SBD). This disambiguation is a non-trivial task because simply splitting text with a small set of punctuation keywords such as `.`, `!`, and `?` is not sufficient. This is because text might also contain quotations that contain these keywords, abbreviations like “Dr.” or simple numbers with decimal points [67]. Moreover, the trailing period of an abbreviation can still mark the end of a sentence [68]. Therefore, more complex algorithms must be used in order to achieve best results.

For the PPE, the text segmentation component utilizes the `StanfordSegmenter` [69] for word segmentation as standalone component in the UIMA pipeline. Furthermore, the `StanfordParser` is responsible for SBD.

6.1.3 Natural Language Parsing

In general, a Natural Language Parser works out the grammatical structure of sentences. It decides which groups of words belong together (form “phrases”) and which words of a sentence are verbs or are the subject or object of a verb [70].

The `StanfordParser` used in this component is a system that, in addition to phrase structure parses, is able to automatically extract typed dependency parses of English sentences. Phrase structures represent the nesting of multi-word constituents, while on the other hand a dependency parse represents dependencies between individual words, such as *subject* and *indirect object* [71].

Such a typed dependency tree can be effectively used to extract textual relations. A graphical representation of the dependencies for the sentence “Include at least one number or uppercase letter.” is illustrated in Figure 6.5. This allows accessing grammatical information such as the predicate-argument structure. Found information is stored in the CAS and can be accessed by proceeding components.

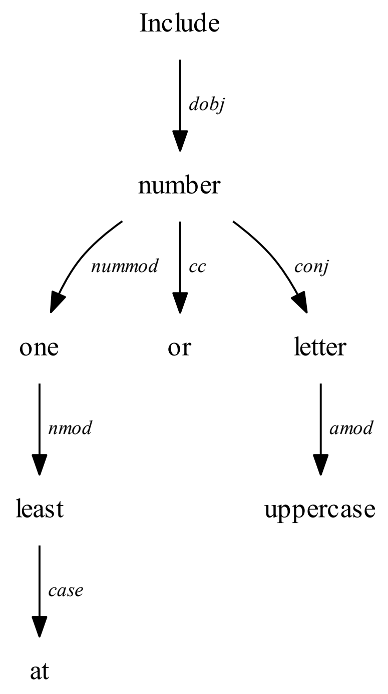


Figure 6.5.: Graphical dependency tree representation of a sample sentence.

6.1.4 Keyword Annotator

The Keyword Annotator consists of a set of Text Analysis Engines (TAEs). They serve the purpose of finding sentences of interest by utilizing a small set of keywords that contain words that may be related to password requirements. In the following, the annotation process is explained in detail.

In the first step, character sets are looked up. The PPE has a fixed set of character sets it is able to recognize, which contains all generally used sets such as:

- Uppercase characters
- Lowercase characters
- Numeric characters
- Special characters

Furthermore, special and more seldom used character sets such as *spaces*, *consecutive characters* or *non-blank characters* can also be recognized.

In the second step, *boundary keywords* are recognized. A boundary keyword is a keyword that defines an upper or lower bound for the use of some object, in this case the character sets. There are boundary keywords that have a slightly different meaning. For example, the requirement “Use at least one lowercase character” and “Use more than one lowercase character” is slightly different due to the different boundary keywords “at least” and “more than”. “More than” states that the number that is referenced by the keyword is not included in the lower bound. The information, whether a boundary keyword includes the provided number or not, is explicitly provided in the keyword set and utilized in the annotation process. This information is stored in the CAS for further processing steps.

In addition to character sets and boundary keywords, the sentences are processed to find certain negations, such as for verbs (e.g. “not use”, “not include”). Also, certain keywords used for specifying a length attribute (e.g. “long”, “length”) help to recognize which sentences are referring to the allowed password length.

6.1.5 Dependency Parsing

In the following, the functioning of the Dependency Parser is described. After keywords have been annotated, the Dependency Parser component uses information to put content into context. More precise, this means that the parser examines the dependencies between different parts of text that are related to password requirements. For example, the parser analyzes whether the presence of a number in a sentence relates to a mention of a character sets. The analysis is performed in a two-step process.

First, ranges of numbers are analyzed. Sentences such as “Use 6-30 characters” require the analysis of numeric ranges. The aim is to recognize the text “6-30” as an allowed range from six to thirty. While the given example could easily be recognized using a simple regular expression, there are more complex structures that should be recognized as well. For instance, the numbers might be written as text string (six, thirty) or as digits as well as text: “Use 6 (six) to 30 (thirty) characters”. For all these cases, the natural language annotations allow the recognition of the dependencies between the numeric values 6 and 30 and therefore the recognition of the numeric range.

Second, numeric ranges and single numerals are put into context with mentions of character sets. Using the dependency tree structure created in Section 6.1.3, the sentence can be analyzed whether the numerals and character sets have a common dependency that forms a relation. Likewise the recognition of dependencies between numerals and character sets, the information from previous processing steps is used to detect dependencies between boundary keywords and character sets as well as other important features such as negations, verbs, and adjectives.

The component does explicitly not *interpret* the found information. This means that it is not known to the component what actual requirements are contained in the sentence. Instead, the component stores found dependencies as new annotations to the CAS to create an annotation structure that can be accessed easily by following components.

6.1.6 Information Interpreter

The Information Interpreter is the final component in natural language processing that takes the information gathered in previous steps and transfers it to an internal data structure representation that expresses password requirements. This data structure can be used to directly create a password policy. Information extraction is done in a recursive algorithm in order to recognize and interpret password requirements from a sentence as a whole. For each sentence, the corresponding dependency tree is traversed and the current node is examined.

The main part forms the recognition of requirements related to character sets. For a node that represents a character sets, dependent requirements such as boundary keywords, numeric ranges or negations are looked up that were recognized in previous processing steps. For a node that represents a numeric value, possible requirements such as boundary keywords and negations are analyzed.

The interpretation of information relates to the process of bringing all information together in order to extract structured information. For example, the requirement “Use not more than 16 characters” has several dependencies that have to be considered. First, the numeral “16” refers to the object “characters”. In combination with the verb “use”, the sentence states that sixteen characters should be used. Considering the additional boundary keywords “more than”, the sentence theoretically indicates that at least seventeen characters must be used. Taking the negation “not” into account, the requirement is reversed. This means, that the Information Interpreter would ultimately find a requirement that enforces a maximum password length of sixteen characters.

The above example gives an impression of the required complexity of the algorithm. There are endless ways only for expressing the maximum length of the password. It also shows that the correct detection and interpretation of boundary keywords is essential. A slightly different boundary keyword can already change the requirement. For instance, the sentence “Do not use 16 or more characters” states a maximum password length of fifteen instead of sixteen.

6.1.7 HTML Meta Information Extraction

The HTML Meta Information Extraction component extracts knowledge directly out of the HTML source code instead of using the preprocessed text. Information about the minimum and maximum length of the password can be found using the password input field present in the document. In contrast to other extracted information, this data is considered as structured information where it is not necessary to utilize complex algorithms for knowledge extraction. In the following, the approach for the extraction of this information is explained.

Apart from unstructured information, a HTML document does also contain (semi-)structured data: The markup itself. Semi-structured data is data that contains tags or other markers that separate semantic elements. The HTML source code contains tags and attributes that serve this purpose. Besides the `placeholder` and `data-*` attributes as explained in Section 6.1.1, the input element allows specifying the `maxlength` attribute [72]. This attribute specifies the maximum number of characters that are allowed to be entered in an input field (see Figure 6.6).

```
<label for="password">
        <strong>Password</strong>
        <input id="password" maxlength="15" name="password" required="required" autocomplete="off" type="password">
</label>
```

Figure 6.6.: Password input with specified maximum length on barnesandnoble.com.

In combination with inputs of the password type, this attribute can be used in order to retrieve an upper bound for the password length. However, it cannot be used in all cases to provide information about the maximum length of a password. In few cases, service providers use the attribute in combination with a textual requirement that states a maximum password length lower than the length stated in the attribute. Therefore, this information is not solely used but in conjunction.

Looking at the HTML specification [73], it seems rather counter-intuitive, but there exists no complementary `minlength` attribute that can be used in order to specify a minimum required length of characters for the provided data in an input element. However, during the development of this software, tests showed that service providers use the minlength attribute as often as the maxlength attribute. Therefore, the component uses both information in order to retrieve upper and lower bounds for the length of the password that must be chosen.

6.1.8 Requirement Storage

The Requirement Storage is the last processing step in the UIMA pipeline and creates the final password policy. The component is responsible for storage of the information that is present in the specialized internal data structure created by the Information Interpreter (cf. Section 6.1.6). In the following, the process of password policy creation is described and issues of the extracted information are discussed.

In the process of finding password requirements, there are cases where ambiguous, duplicate, or conflicting information is found. Before mapping the requirements into the PPML, this information must be cleaned. One example is the previously mentioned information that is found through the extraction of structured information using the HTML markup. This information is not always accurate because it might only provide upper or lower boundaries. Therefore, the information is merged in a way that the extracted requirements through natural language processing is prioritized over HTML markup information. In other cases, it might occur that wrong information was extracted. For instance, the markup information extraction might have selected the wrong password input element and therefore extracted requirements for an additional PIN that has to be used during the login. In this case of such conflicting information, the found information gets discarded.

There are other cases where also multiple information found through natural language parsing is conflicting. For instance, a found requirement might state that there must be no spaces in a password, while another one states that there must be at least one space present. Such information conflicts get resolved through prioritizing one information over another, depending on the conflict of information. For the example of spaces, the exclusion of spaces would be prioritized because it can generally be assumed that this restriction is more likely to occur than the other.

Furthermore, textual password requirements are in general ambiguous. Different service providers use the same text to express different requirements. For instance, looking at the requirement "Use numbers". A service provider might use this statement to tell the user that numbers are allowed to be used in the password. Another service provider might use the same statement in order to express that at least one number must be present in the password. Because this cannot be disambiguated without testing the password, the component sets a minimum occurrence of one for all found character sets that are allowed to be used in the password.

It will be shown in the evaluation (see Section 6.2) that the overall process of extracting password requirements out of a given HTML document into a password policy is applicable to generate secure and valid passwords for the majority of tested services.

6.1.9 Limitations

As the CC, the PPE is also processing English texts only. The implementation of the PPE is focused to extract requirements only related to passwords. Other requirements such as for the username are not extracted, as they also cannot be expressed by the PPML.

The PPML also supports the storage of password management information. This information includes procedures that can be used in order to automatically login to a service, change or reset a user's password as well as information of the expiry period of a password. This password management information is not extracted by the PPE.

Obviously, the PPE is only able to extract requirements that are present in the source code of the given document. There are multiple reasons why a password requirement might not be present in the source code. One of the reasons is that the CC was not able to crawl all requirements, as stated in Section 5.1.5. Another reason might be that the service just does not state all requirements. This is especially the case for requirements on the character sets that can be used for a password.

Regarding the natural language parsing, there is no recognition of referenced objects across multiple sentences. For instance, the keyword “password” might only occur once in a paragraph that has multiple sentences containing requirements related to the password. This limitation is bypassed by the HTML preprocessing that filters requirements that are not related to the password. Without preprocessing, it might otherwise occur that requirements get extracted that are not password related. Nevertheless, the evaluation will show that this limitation only has a small impact on the performance of the PPE.

6.2 Evaluation

In the following, the performance of the PPE will be evaluated. This evaluation is divided in multiple parts. The evaluation set used for the evaluation is presented in Section 6.2.1. Section 6.2.2 explains assumptions that were made about missing information. The correctness criteria explained in Section 6.2.3 explains when a created password policy is considered as correct. Section 6.2.4 presents the evaluation results and concludes this chapter.

6.2.1 Evaluation Set

This section describes the set of documents that were used for evaluation. From the set of domains as described in Section 4, the signup pages of the services were manually looked up and stored. As for the CC, only websites with an available signup page were used. Although it would be possible that the PPE falsely extracts requirements from a document that is no signup page and does not contain password requirements, the impact of this error is low because these policies would never be requested in a real application. Therefore, it is assumed that the component that provides the document for password policy extraction is providing correct content.

The evaluation set contains a variety of different password requirements (see Table 6.1). Besides requirements on the minimum and maximum password length, requirements on character sets such as letters, numbers and special characters are usually used to enforce users to select strong passwords.

Password Requirement	**Occurences**
Minimum Password Length	187
Maximum Password Length	112
Disallowed Identical Consecutive Characters	8
Letters	31
Lowercase Letters	27
Uppercase Letters	32
Numeric Characters	70
Alphanumeric Characters	5
English Characters	1
Special Characters	33
Spaces	24

Table 6.1.: Password requirements present on the evaluation set.

6.2.2 Assumptions

There are several cases in which a service might not provide all information that is necessary in order to create a valid password policy. Because this missing information must still be provided, there are several default values that are used in such cases.

First, a service might not provide information about the allowed character sets. For instance, a requirement might state that the user must "use more than 6 characters". This information does not tell the user which characters are allowed to be used for the password. Therefore, if no allowed character sets are provided, it is assumed that the allowed characters are English lowercase letters, uppercase letters and numbers. For the evaluation set, this was a common set that was accepted by all services. Other characters such as special characters (`!`, `?`, `#`, `%`, ...) are accepted by some services, but not all of them. Similarly, these default character sets will not be considered as not allowed until a requirement is found that explicitly states so. This means a resulting policy for a website with the only requirement such as "Use at least one letter" still allows numbers because they are not explicitly restricted.

Second, other services give no information about the allowed length of the password. In this case, the PPML allows omitting these values in order to leave them as unspecified. Ultimately this means that applications such as password generators will decide about the length of the generated password.

Finally, some services allow or demand the use of special characters. While it is common practice to force users to use certain character sets, all character sets must be specified in the password policy. However, a large amount of websites with the requirement to use special characters do not specify which characters are included in this set. The PPE uses the following characters as special characters, as they represent a character set that was accepted by all services in the evaluation set: `. , : ; - + * ? % & =`

6.2.3 Correctness Criteria

In this section, a definition is provided in which case a password policy is considered as correct. Furthermore, issues and causes of incorrect policies are discussed.

Automating the creation of password policies requires that applications can rely on the information provided by the policies. Otherwise applications like password generators might create invalid passwords that are rejected by the service. As described in Section 6.2.2, in many cases it is not possible for a password policy to exactly reflect a service's password requirements because services often provide incomplete and unclear requirements. For instance, a service might state the characters that are not allowed to be used in a password (e.g. "Do not use spaces"), but does not specify which characters are actually allowed. PPML would still require the password policy to have defined the allowed characters. Furthermore, services might state that special characters are allowed to be used, but do not specify which characters are considered as such. Therefore, it is not demanded that an extracted password policy exactly matches the password requirements of the service. For a password policy to be considered as correct, it is required that the policy can be used in order to create random and secure passwords that will be accepted by the service.

Incorrect information about password requirements, and therefore incorrect policies can cause multiple issues which are discussed in the following. The issues can be distinguished in inconveniences for the users and security threats, i.e. the generation of weak passwords.

1. *Weak Passwords.* Incorrect information about the minimum and maximum password length and/or the character sets can cause the generation of weak passwords that are still accepted by the service. Using a weak password puts the user's account at the service at risk. For instance, a policy might define a maximum length of one instead of ten characters. For services like wikipedia.org, these insecure passwords would still be accepted, because they accept passwords of length one. For evaluation, it is assumed that password generators use the best possible security level when generating passwords and therefore use the maximum length.
2. *Invalid Passwords.* Incorrect password lengths and/or incorrect character sets might cause password generators to generate passwords that do not fulfill the actual requirements of the service and therefore are rejected. For instance, a password policy might state that the password must contain at least one special character, while the service actually allows letters and numbers only. Users would then need to find out the password requirements and configure the password generator manually again.

Therefore, two classifications of correctness are introduced. A policy is exact if all requirements given by the service are extracted correctly and represented in the created password policy. If not all requirements could be extracted, a policy is still considered as correct if it fulfills the following requirements:

- The maximum password length provided in the policy is not significantly (more than 10%) lower than the actual allowed maximum password length.
- For services that do not provide a minimum length, the password length in the password policy must also be unspecified or not lower than 10 characters.
- The password policy does not allow significantly less (more than 10%) of the actual allowed characters.
- A password generator, using the maximum allowed password length[1], will always generate passwords that are accepted by the respective service.

6.2.4 Results

Using the correctness criteria introduced in Section 6.2.3, the PPC successfully provided password policies for 91,5% (183 out of 200) of the evaluated sites. This means that these policies can be used to generate random and valid passwords for the services. The amount of correct policies is splitted in two key figures (see Table 6.2). For a total of 167 (83.5%) of the evaluated services, the PPE correctly extracted all password requirements present in the HTML documents. Furthermore, for 13 (8.0%) of the services, not all password requirements were extracted correctly, but the resulting password policies were still considered as correct because they enforce passwords that are accepted by the service.

[1] If no maximum password length is given by the service, the password generator will choose the length of the password.

Classification	Amount of policies	Percentage
Exact	167	83.5%
Correct	16	8.0%
Erroneous	17	8.5%

Table 6.2.: Evaluation results of the PPE.

There are several reasons why a policy does not exactly match the requirements. The main reason is a different value for minimum/maximum occurrences of character sets (see Table 6.3). For instance, a requirement might say "Use at least one number or special character". The PPC will map these requirements into "At least one number" and "At least one special character". While this does not significantly affect the password security, it does not exactly match the stated requirements. Other reasons are too low maximum password lengths or incorrect minimum password lengths that do not affect the correct generation of passwords. For instance, the PPE extracted a minimum length of six instead of four for the service at bleacherreport.com. For paypal.com, the PPC created a policy that requires the password to contain at least one number *and* symbol, whereas the actual requirement is that the password must contain at least one number *or* symbol.

Cause for imprecise policies	Occurences
Lower maximum length	4
Different minimum length	8
Fewer allowed characters	4
Different character occurrences	14

Table 6.3.: Reasons for imprecise password policies. A single policy may occur multiple times.

The 17 (8.5%) remaining password policies that are considered as incorrect can be further divided into two groups:

- *Inaccurate policies.* These policies enforce the generation of passwords that will in some cases, eventually, be rejected by the service. An example for such an incorrectly crawled password policy is cisco.com: The requirements on the website state that at least one uppercase and one lowercase character must be present in the password. The maximum length of the password is 50 characters. However, the PPC did only extract the requirement on the maximum length of the password. While a password generator, choosing random passwords of length 50 will generate passwords that in most cases contain at least one uppercase and lowercase character, it is still possible that one of these requirements is not met. Therefore, this policy cannot be considered as correct.

- *Incorrect policies.* These policies have requirements that in most (or all) cases allow passwords, that are not accepted by the service. An example for an incorrect policy is the policy for the service at *gamefaqs.com*: The PPC extracted a maximum password length of 64 characters while the actual maximum length is 40. This is due to the fact that the website has two password fields of which one has a maximum length set to 40, while the maximum length of the second one it set so 64. A password generator that always uses the maximum length is never able to generate accepted passwords.

The evaluation shows that the PPE, and therefore the PPC, is able to provide correct password policies for the majority of tested services. Therefore, the PPE can be used in a large-scale environment to generate password policies for a large amount of services. The integration of these password policies in password generators allows an user-friendly and easy-to-use solution for creating random passwords in accordance to a service's requirements.

7 Large-Scale Password Policy Generation

This chapter describes a large-scale crawling for password policies that was conducted as part of this thesis. To solve the issue of missing password policies that can be used to generate passwords in accordance to a service's requirements, the PPC was run on a total of one million domains. In total, the PPC created password policies for 72,125 services.

The PPC was run in the period from October to November 2015. As input, the Quantcast list of the top one million domains [74] were used. This list includes the top ranking domains based on the number of people visiting the websites from the United States. Using the search engine implemented in the CC (cf. Section 5.1.1), URLs to possible signup pages could be found for 406,288 domains. From this list of domains, the websites behind 167,926 were not available (offline) and the pages behind 1,725 domains were not parsable. This sums up to an amount of 236,837 domains that could be crawled by the CC. Using the signup page detection and link extraction, signup pages for a total of 72,125 domains could be found. Each signup page was processed and a corresponding password policy was created. Therefore, the PPC generated 72,125 password policies (see Table 7.1).

Filter	**Websites**	**Remaining Websites**
Total Domains	1,000,000	
No search engine results	593,512	406,488
Service not available	167,926	238,562
Exception while parsing	1,725	236,837
No signup page detected	164,712	**72,125**

Table 7.1.: Overview of filtered domains during the large-scale crawling.

The crawling establishes a database of password policies for a very large amount of domains. Password generators that are able to process password policies can use these policies to generate secure and accepted passwords for the crawled domains. The goal of the large-scale crawling for password policies was to create a large set of usable password policies. Therefore, no further analysis of the security level of these policies has been performed. The future work (cf. Section 9) covers such a security analysis.

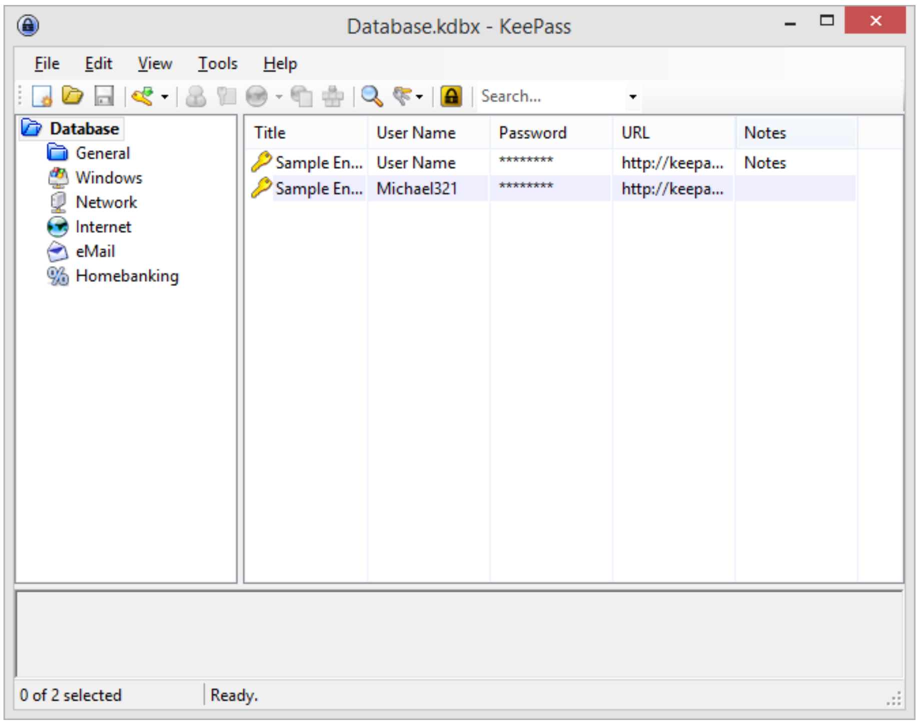
8 Automated Password Generation

This chapter presents a password generator that does not need to be manually configured by the user. Instead, random passwords that are in accordance with a service's requirements can be generated only by providing the URL of the service, for which the password should be generated. The password generator is implemented as extension for the password manager KeePass [8]. It can be used with the current version of KeePass Professional Edition (version 2.30) which allows the integration in an already existing password database.

After a brief introduction into the password management software KeePass and its default functionality for adding account information, the usage of the plugin is presented. The extension was initially developed by the author of this thesis for an earlier version of the PPML [1].

8.1 KeePass

KeePass is a widely used application for password management, which is published under the GNU General Public License (GPL). The application allows the secure storage of account information such as the username, password, the URL of the service, and the expiry of the password. The user interface is illustrated in Figure 8.1. For better transparency, stored account information can be arranged in groups. The right side shows the account entries present in the database for the currently selected group.



Title	User Name	Password	URL	Notes
Sample En...	User Name	********	http://keepa...	Notes
Sample En...	Michael321	********	http://keepa...	

Figure 8.1.: KeePass graphical user interface.

The database containing the account information can be encrypted with Twofish or the Advanced Encryption Standard (AES). AES is considered as secure encryption algorithm [75] which is also used for classified governmental documents [76]. The encryption of the database is not limited to the passwords, but also includes all other account information.

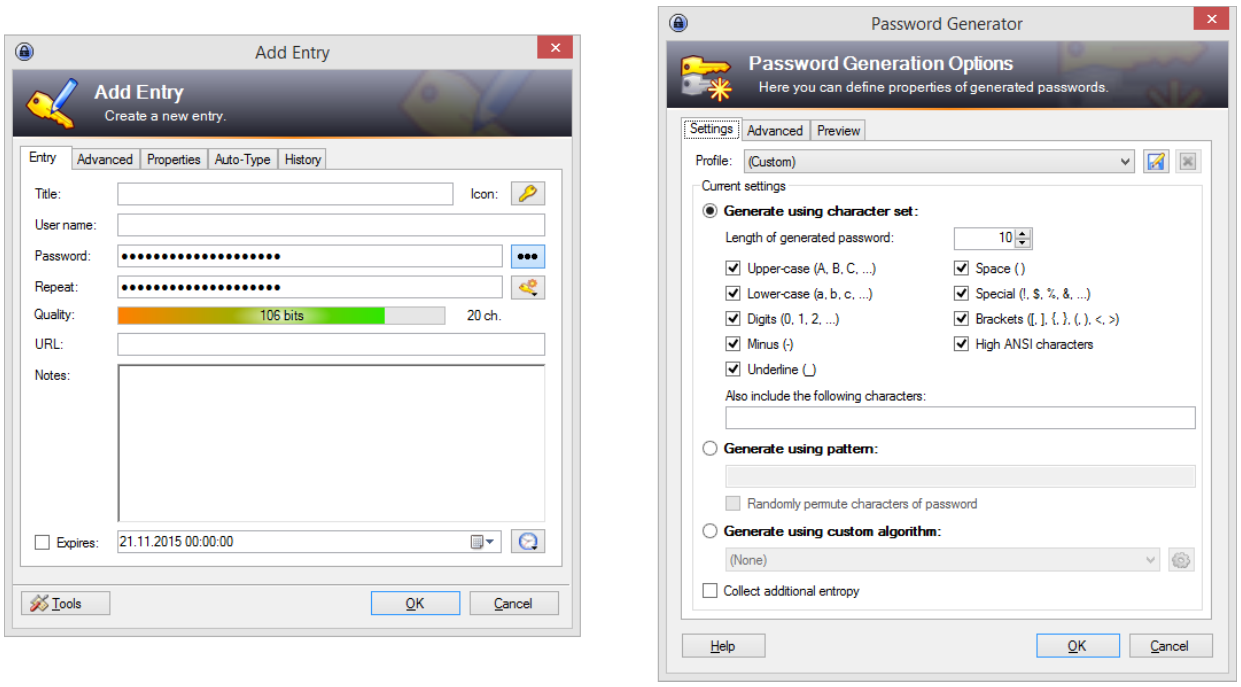
KeePass includes several usability and security features that make this application an easy-to-use and secure password manager. The integrated memory protection ensures that external programs are not able to retrieve passwords from the memory while the encrypted database is opened in KeePass. Using the *Auto-Type* feature, stored account information can be automatically transferred to other applications like a browser in order to automatically log in the user.

KeePass is written in C# and allows adding features by installing extensions. Extensions can add a variety of new features such as import and export functionality, different encryption methods for the database or new password generators.

Password Management

This section describes the default functionality of KeePass for adding account information and generating random passwords. It will be shown that the current functionality for generating passwords it not user-friendly and does not necessarily generate accepted passwords.

Users can add new account information using the *Add Entry* dialog in KeePass. Besides the password, the user can add a wide range of information associated to a service's account. The user can either enter a manually chosen password or utilize the integrated password generator to generate a random password. Random password generation can either be performed using a previously defined configuration, or using the password generator dialog (see Figure 8.2).



Password: ••••••••••••••••••••
Repeat: ••••••••••••••••••••
Quality: 106 bits 20 ch.
Expires: 21.11.2015 00:00:00

Profile: (Custom)
Current settings
(●) Generate using character set:
Length of generated password: 10
[✓] Upper-case (A, B, C, ...) [✓] Space ( )
[✓] Lower-case (a, b, c, ...) [✓] Special (!, $, %, &, ...)
[✓] Digits (0, 1, 2, ...) [✓] Brackets ([,], {, }, (,), <, >)
[✓] Minus (-) [✓] High ANSI characters
[✓] Underline (_)
Also include the following characters:
( ) Generate using pattern:
[ ] Randomly permute characters of password
( ) Generate using custom algorithm:
(None)
[ ] Collect additional entropy

Figure 8.2.: Add Entry dialog in KeePass with opened password generator configuration dialog.

The integrated password generator only has a limited set of options that can be configured. Besides the length of the generated passwords, the allowed characters can be provided using a fixed set of selectable character sets. Additionally, users can manually provide further allowed characters. However, it is obvious that this configuration is not sufficient for all services. Services might have requirements such as "Use at least one uppercase character" that might not necessarily be fulfilled by the random password, especially for short passwords. Furthermore, services may require passwords to not include two identical consecutive characters which cannot be configured. Therefore, users might need to adjust the generated password manually to be accepted by the service.

8.2 Password Generation using Password Policies

This section presents a password generator that is developed as extension for the KeePass password manager. The extension allows the generation of passwords in accordance with a service's password requirements by specifying the URL to the service. This makes the manual configuration of the password generator obsolete.

The extension adds a reduced version of the Add Entry dialog to the application, which allows the specification of all basic account information. The dialog is accessible from the same locations where the original dialog can be accessed. For instance, the user can right click on the user interface to add a new entry in the currently selected group (see Figure 8.3).

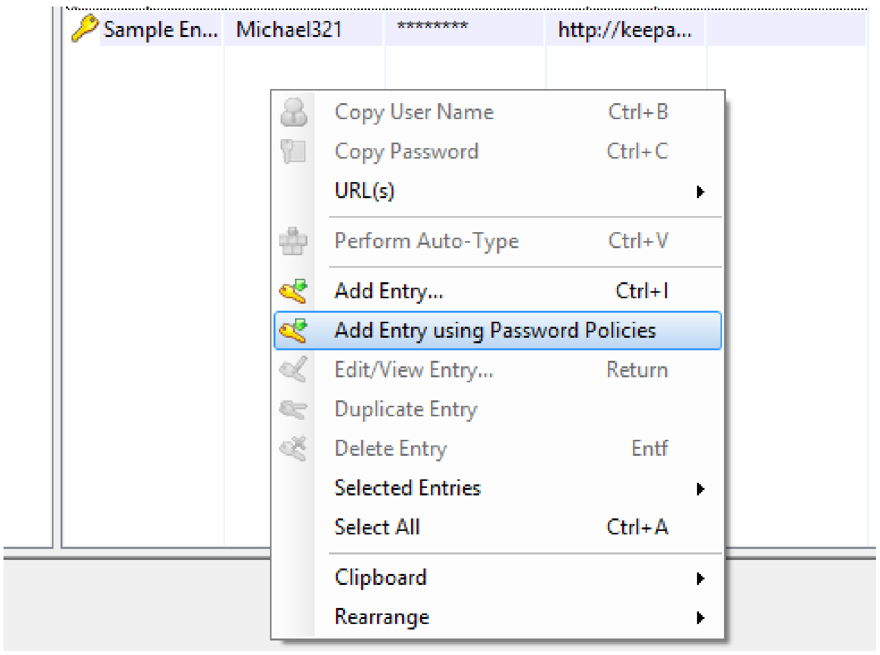
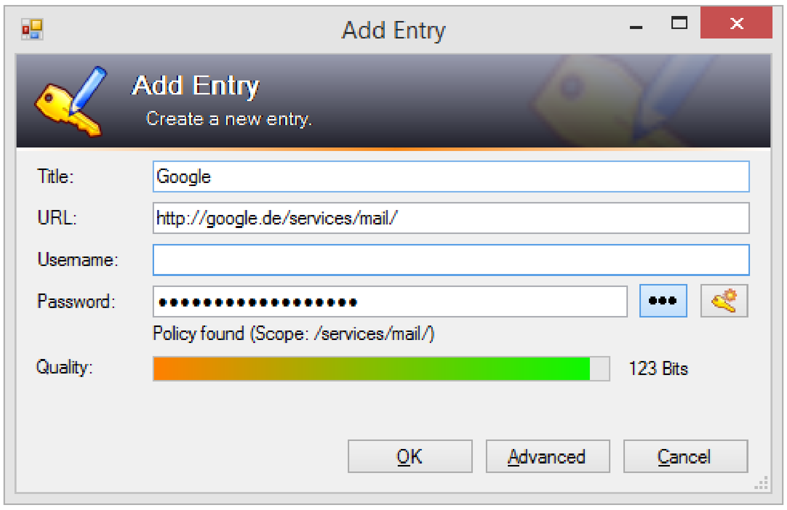


Figure 8.3.: Password generator using password policies integrated in the KeePass context menu.

In the new dialog the user can provide a title for the account entry, the URL to the service, the username, and password (see Figure 8.4). After the URL to the service has been entered, the extension checks whether a password policy can be found that has a scope matching the entered URL. The user can then click the *key icon* to generate a password in accordance to the requirements present in the password policy. The quality meter gives an indication about the strength of the generated password.



Title: Google
URL: http://google.de/services/mail/
Username:
Password: ••••••••••••••••••
Policy found (Scope: /services/mail/)
Quality: 123 Bits

Figure 8.4.: Reduced version of the Add Entry dialog using password policies.

After the password has been generated, the user has two options to save the account information. Clicking OK creates a new entry and stores it in the database. The Advanced button opens the original Add Entry dialog and transfers the already provided information so that the user is able to enter advanced settings such as information regarding the Auto-Type feature.

9 Conclusion

This thesis provided a comprehensive solution for an easy-to-use approach that allows users to generate secure and unique passwords for services on the internet. First, the PPC was introduced. The PPC automatically creates the necessary password policies that had to be created manually until now. Furthermore, the crawling of one million domains provided an initial set of password policies for 72,125 services. With the provided password generator, a comprehensive solution is provided to allow the generation of passwords that are accepted by a service without the need for manual configuration.

It was shown that the PPC achieves good overall results in the process of password policy creation. The first module of the PPC, the CC, achieves a high precision of about 94% and a recall of 74.5% in the process of finding the signup page for a service. The precision indicates that the vast majority of returned pages are correctly classified as signup page. Furthermore, the recall indicates that the CC is able to return signup pages for around 75% of the given domains. The second module, the PPE, uses the crawled signup pages and extracts the password requirements. Using the PPML, these requirements are mapped into a password policy, a machine-readable representation of password requirements. The evaluation showed that 91.5% of the returned password policies were correct and could be used to generate accepted passwords.

The created password policies can be used by applications such as password generators in order to automatically generate accepted passwords without the need for manual configuration of the generator. This eliminates the burden for users having to manually find out the password requirements and configure the password generator when registering for a service. In this thesis, an extension for the password management software KeePass was introduced. The extension provides a password generator that accepts a URL to a service as input. This information is sufficient for the extension to look up the password policy and generate accepted passwords. The 72,125 password policies that were extracted in the large-scale password policy generation can be used to set up a service that provides password generators the necessary password policies.

To conclude, the development of the PPC represents a positive step in providing users an easy-to-use solution for creating secure and accepted passwords for each service. This helps to eliminate security risks that are associated with password reuse and the choosing of weak password. Moreover, the PPC opens up completely new possibilities for the development of applications that use these password policies. Besides password generators, the policies can be used to analyze the enforced password strength of services on the Internet.

Future Work

This section provides future work on multiple topics regarding the PPC and password policies. First, future work on the CC is provided, how the crawling of signup pages can be further improved. Second, a community-based approach for creating missing password policies is discussed. Finally, future work on a security analysis of the crawled password policies is provided.

The CC currently utilizes a search engine in order to find signup pages of a given service. After a signup page has been found, the password input field is focused (cf. Section 5.1.4) and the resulting HTML markup is stored. While this works well for most of the services, there exist some limitations with this approach. The crawler is only able to store requirements that are present at this time of the process. However, some services only show the requirements when an invalid password is entered. For example, a user might type in a password of five characters. After the user removes the focus from the input field, an error message is shown that states that the password must be six or more characters. After the user enters another character and removes the focus from the field, another error is shown stating that the password must have at least one upper case letter. Currently, the CC is not able to interact with websites in a way that these requirements can be crawled. Therefore, it can be extended to interact in a way that such requirements are also stored in the resulting document that is processed by the PPE.

The PPE is able to extract correct password policies for 91.5% of tested documents. To overcome the gap of the missing policies and/or incorrect policies, a community-based approach can be used. While it would not be applicable to solely have a community-based approach because of the huge number of services on the internet, it can be used complementary. In a community-based approach, users could submit new password policies that did not exist before, for instance for new services. The requirements could be entered in a graphical user interface that allows creation of password policies without the need of writing pure XML files. Additionally, users could use this tool to correct existing password policies. Besides the possibility that a policy is incorrect because of an erroneous requirement extraction, it may also be outdated because a service updated its requirement. Both issues can be addressed with a community-based approach in a centralized environment that provides password policies.

The large-scale password policy generation provided a database of 72,125 password policies. The goal of the generation of password policies was to have a large set of policies that can be used by the developed password generator. The set of password policies can be used to analyze the password security level of the crawled websites. For example, an average minimum and maximum password length can be derived from the data. Calculating a maximum and minimum entropy possible for the generation of passwords can help to classify services with regard to password security. Furthermore, the set can be used to analyze whether it is possible to create a password policy that can be used to generate accepted passwords for the majority of services.

10 Glossary

Analysis Engine

An analysis engine is a component responsible for analyzing unstructured information. An analysis engine contains one or more annotators that include logic for analysis. The analysis logic contained in the annotators process the document and create meta data about artifacts that will be stored in the Common Analysis Structure.

Common Analysis Structure

The Common Analysis Structure (CAS) contains the documents to be analyzed. Analysis Engines add objects to the CAS containing analysis results. As the CAS gets passed along the components in the analysis process, the CAS is more and more enriched with information. The results added by an analysis engine can be accessed and used by all proceeding analysis engines.

Document Object Model (HTML)

The Document Object Model (DOM) in HTML refers to a convention for representation and interaction with objects. The objects are represented as nodes and organized in a tree structure, called the DOM tree. Using the DOM, the structure of the document can be traversed and objects can be added, modified and deleted. The objective of the DOM is to provide a programming interface that can be used in a wide range of applications.

Sentence Boundary Disambiguation

Sentence Boundary Disambiguation (SBD) is the process of dividing a document text into sentences. This disambiguation is a non-trivial task, because simply splitting text using punctuation keywords such as `.`, `!`, and `?` is not sufficient. For example, a text might contain abbreviations like “Dr.” or numbers with decimal points that do not mark the end of a sentence. Furthermore, question marks and exclamation marks may appear in quotations without marking the end of a sentence. Toolkits such as Apache OpenNLP [77] contain complex algorithms that are able to split over 95% of sentences correctly.

Structured information

In Natural Language Processing, structured information refers to information that is structured in fields such as “title”, “price” and “stock”. Structured information is often stored in relational databases. This allows querying the database to answer questions like “How many products are out of stock?”.

Tokenization

Tokenization refers to the process of breaking a document text into meaningful elements called tokens. Tokens can be of different types such as words, phrases or sentences. For instance, word segmentation takes a document texts and breaks it into words. For the English language, this task does not need complex algorithms, because there exist orthographic spaces between words. The Sentence Boundary Disambiguation is a special case of tokenization.

Unstructured Information

The term unstructured information (or *unstructured data*) refers to content that is not organized in a pre-defined data model. Unstructured information typically includes text or multimedia content such as emails, presentations and webpages. In contrast to structured information, such content cannot be fit into a relational database that can be used to gain information.

Bibliography

[1] Mario Schlipf. Passwort-Richtlinien. Bachelor Thesis, TU Darmstadt, October 2014. `http://www.cdc.informatik.tu-darmstadt.de/reports/reports/Mario_Schlipf.bachelor.pdf`. (I, II, 2, 41)

[2] Xiaoyuan Suo, Ying Zhu, and G Scott Owen. Graphical passwords: A survey. In *Computer security applications conference, 21st annual*, pages 10–pp. IEEE, 2005. (2)

[3] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *Symposium on Network and Distributed System Security (NDSS)*, 2014. (2)

[4] Blake Ives, Kenneth R Walsh, and Helmut Schneider. The domino effect of password reuse. *Communications of the ACM*, 47(4):75–78, 2004. (2)

[5] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007. (2)

[6] LastPass. LastPass Password-Manager, 2015. `https://lastpass.com/`. (2)

[7] Dashlane. Password Generator - Generate random passwords, 2015. `https://www.dashlane.com/password-generator`. (2)

[8] D. Reichl. KeePass Password Safe, 2015. `http://www.keepass.info`. (2, 41)

[9] RANDOM.ORG Ltd. RANDOM.ORG Password Generator, 2015. `https://www.random.org/passwords/`. (2, 4)

[10] Christian Thoeing. PWGen, 2015. `http://pwgen-win.sourceforge.net`. (2, 4)

[11] Richard Shay, Abhilasha Bhargav-Spantzel, and Elisa Bertino. Password policy simulation and analysis. In *Proceedings of the 2007 Workshop on Digital Identity Management, Fairfax, VA, USA, November 2, 2007*, pages 1–10, 2007. (4)

[12] Anna C Squicciarini, Abhilasha Bhargav-Spantzel, Elisa Bertino, and Alexei B Czeksis. Auth-SL-a system for the specification and enforcement of quality-based authentication policies. In *Information and Communications Security*, pages 386–397. Springer, 2007. (4)

[13] Richard Shay and Elisa Bertino. A comprehensive simulation tool for the analysis of password policies. *Int. J. Inf. Sec.*, 8(4):275–289, 2009. (4)

[14] Password Requirements. `http://passrequirements.com`. (4)

[15] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. Does My Password Go up to Eleven? The Impact of Password Meters on Password Selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2379–2388. ACM, 2013. (4)

[16] Craigslist, 2015. `http://craigslist.com`. (4)

[17] David Recordon and Drummond Reed. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. ACM, 2006. (5)

[18] Moo Nam Ko, Gorrell P Cheek, Mohamed Shehab, and Ravi Sandhu. Social-networks connect services. *Computer*, (8):37–43, 2010. (5)

[19] Manuel Urueña, Alfonso Muñoz, and David Larrabeiti. Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites. *Multimedia Tools and Applications*, 68(1):159–176, 2014. (5)

[20] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 365–379. IEEE, 2012. (5)

[21] Xavier de Carné de Carnavalet and Mohammad Mannan. From Very Weak to Very Strong: Analyzing Password-Strength Meters. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014. (7)

[22] Waraporn Viyanon. Structure and content semantic similarity detection of eXtensible markup language documents using keys. 2010. (8)

[23] Eric van der Vlist. xsd:dateTime Datatype Reference, 2015. `http://books.xmlschemata.org/relaxng/ch19-77049.html`. (9)

[24] Graham Klyne and Chris Newman. Date and time on the internet: Timestamps. 2002. (9)

[25] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Methods for domain-independent information extraction from the web: An experimental comparison. 2004. (10)

[26] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005. (10)

[27] P Ana-Maria. Information extraction from unstructured Web text [Ph. D Dissertation]. *University of Washington*, 2007. (10)

[28] David Ferrucci and Adam Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004. (10, 11, 27)

[29] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003. (10)

[30] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986. (10)

[31] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. (10)

[32] Larry Wos, Ross Overbeck, Ewing Lusk, and Jim Boyle. Automated reasoning: introduction and applications. 1984. (10)

[33] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003. (10)

[34] Alexa. Top Sites in United States, 2015. `http://www.alexa.com/topsites/countries/US`. (13)

[35] Comcast. Xfinity official site, 2015. `http://www.xfinity.com/`. (13)

[36] Jennifer A Bartlett. Internet Reviews: Alternatives to Google. 2014. (15)

[37] StartPage. Search results StartPage web search, 2015. `https://startpage.com/do/search?prf=fda1640a2945756371defed40e240d42&cat=web&query=facebook.com+sign+up`. (15)

[38] World Wide Web Consortium et al. HTML5 specification. *Technical Specification, Jun*, 24:2010, 2010. (15)

[39] Andreas Bruns, Andreas Kornstädt, and Dennis Wichmann. Web application tests with selenium. *Software, IEEE*, 26(5):88–91, 2009. (16)

[40] Thomas Powell. *Ajax: The Complete Reference*. McGraw-Hill, Inc., 2008. (16)

[41] Danny Sullivan. How search engines work. *SEARCH ENGINE WATCH, at http://www. searchenginewatch. com/webmasters/work. html (last updated June 26, 2001)(on file with the New York University Journal of Legislation and Public Policy)*, 2002. (16)

[42] Wikipedia, the free encyclopedia. Create account, 2015. `https://en.wikipedia.org/w/index.php?title=Special:UserLogin&type=signup`. (16)

[43] Martijn Koster. *A standard for robot exclusion*. NEXOR., 1994. (17)

[44] Mark Nottingham and Eran Hammer-Lahav. Defining well-known uniform resource identifiers (URIs). 2010. (17)

[45] Yang Sun, Ziming Zhuang, and C Lee Giles. A large-scale study of robots.txt. In *Proceedings of the 16th international conference on World Wide Web*, pages 1123–1124. ACM, 2007. (17)

[46] iStockPhoto. Sign Up Pictures, Images, and Stock Photos - iStock, 2015. `http://www.istockphoto.com/photos/sign+up`. (17)

[47] Walmart. Create your account, 2015. https://www.walmart.com/account/signup?returnUrl=%2Faccount%2F. (18)

[48] Walmart. JavaScript resource, 2015. https://i5.walmartimages.com/dfw/63fd9f59-90f9/k2-_377fb962-2829-491c-ab4b-e3ae5e03fd8a.v1.js. (18)

[49] Wei Koong Chai, Ning Wang, Ioannis Psaras, George Pavlou, Chaojiong Wang, Gerardo García De Blas, Francisco Javier Ramon-Salguero, Lei Liang, Spiros Spirou, Andrzej Beben, et al. Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services. *Communications Magazine, IEEE*, 49(3):112–120, 2011. (18)

[50] Pandora. Restricted access for Germany, 2015. http://www.pandora.com/restricted. (19)

[51] Facebook. Sign up for Facebook, 2015. https://www.facebook.com/r.php. (20)

[52] Facebook. What is the minimum password strength and how can I make my password strong?, 2015. https://www.facebook.com/help/124904560921566. (21)

[53] Adobe. Create an Adobe ID, 2015. https://accounts.adobe.com/. (23)

[54] CNBC, 2015. http://www.cnbc.com. (23)

[55] T-Mobile. T-Mobile ID, 2015. https://account.t-mobile.com/oauth2/v1/controller. (23)

[56] David Lewis. Representation quality in text classification: An introduction and experiment. In *Proc. Workshop on Speech and Natural Language. Morgan Kaufmann, Hidden Valley, PA*, pages 288–295, 1990. (24)

[57] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006. (24)

[58] Techcrunch. Report: More Than 250M Domain Names Have Now Been Registered, Almost Half Are .Com And .Net, 2013. http://techcrunch.com/2013/04/08/internet-passes-250m-registered-top-level-domain-names/. (25)

[59] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 2011. (26)

[60] W3C. HTML/Elements/select, 2015. https://www.w3.org/wiki/HTML/Elements/select. (27)

[61] W3Schools. HTML input placeholder Attribute, 2015. http://www.w3schools.com/tags/att_input_placeholder.asp. (28)

[62] W3Schools. HTML Global data-* Attributes, 2015. http://www.w3schools.com/tags/att_global_data.asp. (28)

[63] Ancestry. Sign In, 2015. https://secure.ancestry.com/login. (28)

[64] Disney. Create Your Disney Account, 2015. https://register.go.com/global/login. (28)

[65] W3Schools. HTML Block and Inline Elementst, 2015. http://www.w3schools.com/html/html_blocks.asp. (29)

[66] James H Martin and Daniel Jurafsky. Speech and language processing. *International Edition*, 2000. (29)

[67] Jeffrey C Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19. Association for Computational Linguistics, 1997. (30)

[68] Haoyi Wang and Yang Huang. Bondec–A Sentence Boundary Detector. *CS224N Project, Stanford*, 2003. (30)

[69] The Stanford Natural Language Processing Group. Stanford Word Segmenter, 2015. http://nlp.stanford.edu/software/segmenter.shtml. (30)

[70] The Stanford Natural Language Processing Group. The Stanford Parser: A statistical parser, 2015. http://nlp.stanford.edu/software/lex-parser.shtml. (30)

[71] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006. (30)

[72] W3Schools. HTML input maxlength Attribute, 2015. http://www.w3schools.com/tags/att_input_maxlength.asp. (33)

[73] Dave Raggett, Arnaud Le Hors, Ian Jacobs, et al. HTML 4.01 Specification. *W3C recommendation*, 24, 1999. (33)

[74] Quantcast. Top Ranking International Websites, 2011. https://www.quantcast.com/top-sites. (40)

[75] Herman Isa, Iskandar Bahari, Hasibah Sufian, and Muhammad Reza Z'aba. AES: current security and efficiency analysis of its alternatives. In *Information Assurance and Security (IAS), 2011 7th International Conference on*, pages 267–274. IEEE, 2011. (42)

[76] Annabelle Lee. Guideline for Implementing Cryptography in the Federal Government. Technical report, DTIC Document, 1999. (42)

[77] J Baldridge and T Morton. OpenNLP, 2004. (47)

A Password Policy Crawler Manual

This chapter serves to explain the software usage of the PPC. The PPC is a software package that is designed to run in parallel on multiple machines. The ability to split time consuming and computationally expensive tasks of the PPC onto multiple machines makes it necessary to run the PPC sequentially in different run modes. In the following, the necessary steps are explained in order to crawl password policies form a given database of domains.

Retrieving search results

The first step in crawling password policies is to run the CC in the `search` mode. This retrieves search results for a given database of domains. The command looks as follows:

```
java -jar crawler.jar search <target> <source>
```

The `<source>` parameter specifies a path to a database that is expected to be a valid SQLite database. The database must contain a table called `services`. This table must contain the columns `name` and `url` specifying the name and domain of the service. A corresponding table can be created using the following sql statement:

```
CREATE TABLE services (name TEXT, url TEXT);
```

The `<target>` parameter specifies a path to another SQLite database containing a table called `results`. This table is expected to have the columns `ID`, `domain`, `time`, `result1`, `result2`, `result3`. If the given target file does not exist, a database containing the necessary table is automatically created. Otherwise, the table can also be manually created using the following statement:

```
CREATE TABLE results (
        'ID'          INTEGER PRIMARY KEY AUTOINCREMENT,
        'domain'      TEXT UNIQUE,
        'time'        INTEGER,
        'result1'     TEXT,
        'result2'     TEXT,
        'result3'     TEXT
);
```

Crawling Signup Pages

The second step is the crawling of signup pages. In order to do so, the CC is run by using the following command:

```
java -jar crawler.jar crawler <database> <threadCount>
```

The <database> parameter is specifying a path to a database as created in the previous step. The <threadCount> parameter specifies with how many threads the crawling is performed. Each thread is representing the concurrent crawling of found URLs for one domain.

The output in this step is an output folder that is named after the database name. Inside the output folder, subfolders are created for each crawled domain. These subfolders contain the found signup page as well as a log file with debug information. The CC can run with the given amount of threads. However, it is not supported to run multiple instances of the application.

Building the Files List

In the third step, a files list is created that contains the paths to all found signup pages. In order to create the list, the PPE is run using the following command:

```
java -jar extractor.jar build-fileslist <input> <output>
```

The <input> parameter specifies the path to a folder that should be read. The folder is recursively read and can therefore contain output folders from multiple runs of the CC. The <output> specifies the file to which the files list should be written.

Parsing the Signup Pages

The fourth step runs the extraction of password requirements. This step creates the final password policies:

```
java -jar extractor.jar run-parser <filesList>
```

The PPE uses the path to the files list given in the <filesList> parameter to retrieve the paths to the signup pages. The extractor runs in a single threaded environment parsing one signup page at a time. The application uses file based locks in order to check whether another instance is parsing a file contained in the files list. This enables the application to be run multiple times and even on different machines sharing the same storage (e.g. network storage).

The extracted password policies are stored in the file system with the same file name as the signup page including an appended `.xml`.

Utilities

The previously explained four steps are sufficient to create password policies from a existing database of domains. However, there are a few more *utility* run modes that can be used, e.g. for quickly copying the created password policies to another location. In the following, the available utility run modes are briefly described:

1. `java -jar extractor.jar copy-results <input> <output>`

   Recursively copies the created password policies present in the <input> folder to the specified target <output> location.

2. `java -jar extractor.jar clean-locks <path>`

   Recursively deletes all locks created during the process of password policy extraction. This mode should only be run if all parsing processes are finished. Otherwise, already processed documents would be processed again.

3. `java -jar extractor.jar clean-all <path>`

   Additionally to the `clean-locks` run mode, the created password policies will also be deleted. This deletes all files created in the process of password policy extraction.

B PPML XML Schema Definition

```
<?xml version="1.0" encoding="UTF−8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:vc="http://www.w3.org/2007/XMLSchema−versioning" elementFormDefault="qualified"
    xmlns:xerces="http://xerces.apache.org" vc:minVersion="1.1">
    <xs:element name="policies" type="Policies"> </xs:element>
    <xs:complexType name="Policies">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="policy" type="Policy">
                <xs:unique name="uniqueScope">
                    <xs:selector xpath="policy"/>
                    <xs:field xpath="scope"/>
                </xs:unique>
                <xs:key name="characterSetKey">
                    <xs:selector xpath="./characterSets/characterSet"/>
                    <xs:field xpath="@name"/>
                </xs:key>
                <xs:keyref name="requirementRuleKeyRef" refer="characterSetKey">
                    <xs:selector
                        xpath="./properties/characterSettings/requirementGroup/requirementRule"/>
                    <xs:field xpath="@characterSet"/>
                </xs:keyref>
                <xs:keyref name="positionRestrictionKeyRef" refer="characterSetKey">
                    <xs:selector xpath="./properties/characterSettings/positionRestriction"/>
                    <xs:field xpath="@characterSet"/>
                </xs:keyref>
                <xs:keyref name="characterSetSettingsKeyRef" refer="characterSetKey">
                    <xs:selector xpath="./properties/characterSettings/characterSet"/>
                    <xs:field xpath="@name"/>
                </xs:keyref>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Policy">
        <xs:annotation>
            <xs:documentation>The Policy represents a single password policy for the given scope.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="characterSets" minOccurs="0" type="CharacterSets">
                <xs:unique name="characterSetUniqueName">
                    <xs:selector xpath=".//characterSet"/>
                    <xs:field xpath="@name"/>
                </xs:unique>
            </xs:element>
            <xs:element name="properties" minOccurs="0" type="Properties">
                <xs:annotation>
                    <xs:documentation>Required. Represents the properties of the password.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="service" minOccurs="0" type="Service">
                <xs:annotation>
                    <xs:documentation>Holds information related to the service the password is used for.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="version" type="xs:string">
            <xs:annotation>
                <xs:documentation>The current version of the policy.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="timestamp" type="xs:dateTime">
```

```xml
            <xs:annotation>
                <xs:documentation>Timestamp when this policy was created/updated. It must include the time, the date, and the offset from the UTC
                        time.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="scope" type="xs:anyURI" use="required">
            <xs:annotation>
                <xs:documentation>Relative path of the webpage where this policy will be used.
                        If the scope attribute references a file (i.e. does not end with a slash), the policy is only valid for the referenced file. A policy with a scope that ends with
                                a slash is valid for the referenced folder, all subfiles and folders. Policies with a more specific scope are chosen over ones with more general
                                scopes. The default value of the scope is "/", which is valid for all files and folders.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="redirect" type="xs:anyURI"/>
        <xs:attribute name="name" type="xs:string"/>
        <xs:assert
            test="(@redirect␣and␣not(characterSets)␣and␣not(properties)␣and␣not(service))␣or␣(not(@redirect)␣and␣characterSets)"
            xerces:message="Either␣the␣redirect␣attribute␣or␣both␣characterSets␣and␣properties␣elements␣must␣be␣present,␣but␣not␣both."
        />
    </xs:complexType>
    <xs:complexType name="CharacterSets">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="characterSet" type="CharacterSet"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CharacterSet">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="base" type="xs:string">
                <xs:annotation>
                    <xs:documentation>Reference to an already defined character set. All characters from the character set referenced by this element will be added to the
                            character set this element is a child of.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="characters" type="xs:string" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>String containing all characters that will be added to the character set.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
    <xs:complexType name="Properties">
        <xs:all>
            <xs:element minOccurs="0" name="characterSettings" type="CharacterSettings">
                <xs:annotation>
                    <xs:documentation>Parent node for character settings. If the node is omitted, all characters defined in the characterSets element are treated as available
                            for use with no restrictions on minimum and maximum ocurrences.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element default="0" minOccurs="0" name="maxConsecutive" type="xs:nonNegativeInteger">
                <xs:annotation>
                    <xs:documentation>Indicates whether consecutive characters are allowed or not. A omitted value or 0 inidaces no limitation on consecutive
                            characters.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="minLength" type="xs:nonNegativeInteger" minOccurs="0" default="1">
                <xs:annotation>
                    <xs:documentation>Minimum length of the password.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="maxLength" type="xs:nonNegativeInteger" minOccurs="0" default="0">
                <xs:annotation>
                    <xs:documentation>Maximum length of the password. A value of 0 means no maximum length.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="expires" type="xs:nonNegativeInteger" minOccurs="0" default="0"
                nillable="false">
                <xs:annotation>
                    <xs:documentation>Password expiry in days. A value of 0 means no expiry.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="CharacterSettings">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="1" name="characterSet"
                type="CharacterSetSettings">
                <xs:annotation>
                    <xs:documentation>Settings element for a globally available character set.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="requirementGroup"
                type="RequirementGroup">
                <xs:annotation>
                    <xs:documentation>Character sets specified in the requirement groups are implicitly added to the available character sets for the given position (or all
                            position if no positions are specified) if they were not allowed previously. For a given character position this means that first, the available characters
                            are evaluated using the characterSet element and the positionRestriction element. After this, the requirementRules add possible characters, they do
                            not remove previously allowed characters (like the positionRestriction element does).</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="positionRestriction"
                type="PositionRestriction">
                <xs:annotation>
                    <xs:documentation>Restriction element used to restrict the allowed characters for a given character position. The globally available character sets are not
                            available for any character position that has at least one position restriction as long as they are not made available by a positionRestriction
                            element.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CharacterSetSettings">
        <xs:all>
            <xs:element minOccurs="0" name="minOccurs" type="xs:nonNegativeInteger" nillable="false">
                <xs:annotation>
                    <xs:documentation>Minimum password global occurrences of the character set. Omitted for no restrictions on minimum
                            occurences.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element minOccurs="0" name="maxOccurs" type="xs:nonNegativeInteger" nillable="false">
                <xs:annotation>
                    <xs:documentation>Maximum password global occurrences of the character set. Omitted for no restrictions on maximum
                            occurences.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:all>
        <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="PositionRestriction">
        <xs:all>
            <xs:element name="positions" type="xs:string">
                <xs:annotation>
                    <xs:documentation>Comma separated list of character positions the restriction is applied to. Each position can be a character position starting with 0.
                            Negative character positions can be used to specify the position beginning from the end of the password. A value in the interval (0,1) can be used to
                            specify a position by ratio. E.g. 0.5 refers to the center position of the password.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element default="0" minOccurs="0" name="minOccurs" type="xs:nonNegativeInteger">
                <xs:annotation>
                    <xs:documentation>Minimum occurences of the character set for the given positions. A value of 0 means no restrictions of minimum
                            occurences.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element default="0" minOccurs="0" name="maxOccurs" type="xs:nonNegativeInteger">
                <xs:annotation>
                    <xs:documentation>Maximum occurences of the character set for the given positions. A value of 0 means no restrictions of maximum
                            occurences.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:all>
        <xs:attribute name="characterSet" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="RequirementGroup">
        <xs:sequence>
            <xs:element default="1" minOccurs="0" name="minRules" type="xs:positiveInteger">
                <xs:annotation>
                    <xs:documentation>Minimum number of rules that must be fulfilled.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element maxOccurs="unbounded" name="requirementRule" type="RequirementRule"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RequirementRule">
        <xs:all>
            <xs:element default="0" minOccurs="0" name="minOccurs" type="xs:nonNegativeInteger">
                <xs:annotation>
                    <xs:documentation>Minimum occurrences of the given character set. A value of 0 means no minimum occurrences.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element default="0" minOccurs="0" name="maxOccurs" type="xs:nonNegativeInteger">
                <xs:annotation>
                    <xs:documentation>Maximum occurrences of the given character set. A value of 0 means no maximum occurrences.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element minOccurs="0" name="positions" type="xs:string">
                <xs:annotation>
                    <xs:documentation>List of character positions this rule applies to as defined in the PositionRestriction type.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:all>
        <xs:attribute name="characterSet" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="Service">
        <xs:all>
            <xs:element minOccurs="0" name="login" type="ServiceLogin"/>
            <xs:element minOccurs="0" name="register" type="ServiceRegister"/>
            <xs:element minOccurs="0" name="passwordChange" type="ServicePasswordChange"/>
            <xs:element minOccurs="0" name="passwordReset" type="ServicePasswordReset"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ServiceLogin">
        <xs:all>
            <xs:element name="url" type="xs:anyURI" minOccurs="0"/>
            <xs:element minOccurs="0" name="maxTries" type="xs:positiveInteger"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ServiceRegister">
        <xs:all>
            <xs:element name="url" type="xs:anyURI" minOccurs="0"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ServicePasswordChange">
        <xs:all>
            <xs:element name="url" type="xs:anyURI" minOccurs="0"/>
            <xs:element minOccurs="0" name="maxTries" type="xs:positiveInteger"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ServicePasswordReset">
        <xs:all>
            <xs:element name="url" type="xs:anyURI" minOccurs="0"/>
            <xs:element minOccurs="0" name="maxTries" type="xs:positiveInteger"/>
        </xs:all>
    </xs:complexType>
</xs:schema>
```

Listing B.1: Password Policy Markup Language XML Schema

C Example Password Policy

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies>
    <policy scope="https://100mb.co.in/">
        <characterSets>
            <characterSet name="Uppercase">
                <characters>ABCDEFGHIJKLMNOPQRSTUVQXYZ</characters>
            </characterSet>
            <characterSet name="Lowercase">
                <characters>abcdefghijklmnopqrstuvwxyz</characters>
            </characterSet>
            <characterSet name="Numbers">
                <characters>0123456789</characters>
            </characterSet>
            <characterSet name="Special">
                <characters><![CDATA[.,:;-+*?%&=]]></characters>
            </characterSet>
            <characterSet name="Letters">
                <characters>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVQXYZ</characters>
            </characterSet>
        </characterSets>
        <properties>
            <characterSettings>
                <characterSet name="Uppercase">
                    <minOccurs>1</minOccurs>
                </characterSet>
                <characterSet name="Lowercase">
                    <minOccurs>1</minOccurs>
                </characterSet>
                <characterSet name="Numbers">
                    <minOccurs>1</minOccurs>
                </characterSet>
                <characterSet name="Special">
                    <minOccurs>1</minOccurs>
                </characterSet>
                <characterSet name="Letters"/>
            </characterSettings>
            <minLength>8</minLength>
            <maxLength>32</maxLength>
        </properties>
    </policy>
</policies>
```

Listing C.1: Password Policy crawled by the PPC.

D PPC Evaluation

google.com	yelp.com	mint.com	opentable.com	popads.net
amazon.com	zillow.com	steamcommunity.com	myway.com	ksl.com
wikipedia.org	wordpress.com	creditkarma.com	tvguide.com	seekingalpha.com
ebay.com	groupon.com	dailymotion.com	bankrate.com	foxsports.com
twitter.com	cnet.com	telegraph.co.uk	zendesk.com	speedtest.net
craigslist.org	stackoverflow.com	hootsuite.com	qualtrics.com	fanfiction.net
linkedin.com	adobe.com	monster.com	sourceforge.com	investopedia.com
paypal.com	bleacherreport.com	evernote.com	fool.com	macrumors.com
apple.com	twitch.tv	alibaba.com	sharesale.com	gamespot.com
target.com	goodreads.com	match.com	cars.com	disqus.com
outbrain.com	tmz.com	gamefaqs.com	biblegateway.com	wnd.com
usps.com	woot.com	rottentomatoes.com	bizjournals.com	fatwallet.com
ups.com	cbssports.com	marketwatch.com	cafemom.com	basecamp.com
forbes.com	engadget.com	bedbathandbeyond.com	starbucks.com	jcrew.com
slickdeals.net	surveymonkey.com	qvc.com	thefreedictionary.com	arstechnica.com
lowes.com	allrecipes.com	nypost.com	refinery29.com	liveleak.com
fedex.com	godaddy.com	hotels.com	primewire.ag	cisco.com
kohls.com	nydailynews.com	fandango.com	stumbleupon.com	cc.com
guthub.com	united.com	barnesandnoble.com	livestrong.com	victoriassecret.com
newegg.com	kickstarter.com	bhphotovideo.com	officedepot.com	uproxx.com
wsj.com	aa.com	photobucket.com	vox.com	espncricinfo.com
expedia.com	weebly.com	ebates.com	battle.net	timeanddate.com
theguardian.com	ticketmaster.com	wix.com	orbitz.com	businessinsider.com
wunderground.com	coupons.com	webex.com	rei.com	zulily.com
southwest.com	jcpenny.com	shutterfly.com	shopify.com	meetup.com
nordstrom.com	houzz.com	hilton.com	pogo.com	eventbrite.com
gap.com	zappos.com	kbb.com	drugs.com	nfl.com
costco.com	delta.com	babycenter.com	gamestop.com	lifebuzz.com
ancestry.com	theverge.com	instructables.com	xda-developers.com	deviantart.com
staples.com	myfitnesspal.com	addthis.com	thechive.com	dailymail.co.uk
npr.org	mailchimp.com	barclaycardus.com	kmart.com	ign.com
accuweather.com	dell.com	adcash.com	aarp.org	retailmenot.com
time.com	wayfair.com	slideshare.net	shutterstock.com	mapquest.com
constantcontact.com	airbnb.com	forever21.com	angieslist.com	yellowpages.com
okcupid.com	hp.com	evite.com	livejournal.com	cnbc.com
go.com	fitbit.com	topix.com	tigerdirect.com	priceline.com
imdb.com	whitepages.com	bodybuilding.com	samsung.com	change.org
nytimes.com	sbnation.com	reuters.com	zergnet.com	city-data.com
walmart.com	t-mobile.com	careerbuilder.com	stubhub.com	signupgenius.com
weather.com	theblaze.com	archive.org	tomshardware.com	howtogeek.com

Table D.1.: Domains used for evaluating the PPC modules.

Domain	Found using Search Engine	Found using Link Extraction	Signup Page returned	Returned Signup Page correct
google.com	Yes	-	Yes	Yes
amazon.com	No	Yes	Yes	Yes
wikipedia.org	No	Yes	Yes	Yes
ebay.com	Yes	-	Yes	Yes
twitter.com	Yes	-	Yes	Yes
craigslist.org	No	No	No	-
linkedin.com	Yes	-	Yes	Yes
paypal.com	Yes	-	Yes	Yes
apple.com	Yes	-	Yes	Yes
target.com	No	No	No	-
outbrain.com	Yes	-	Yes	Yes
usps.com	Yes	-	Yes	Yes
ups.com	No	No	No	-
forbes.com	Yes	-	Yes	Yes
slickdeals.net	Yes	-	Yes	Yes
lowes.com	Yes	-	Yes	Yes
fedex.com	Yes	-	Yes	Yes
kohls.com	Yes	-	Yes	Yes
github.com	Yes	-	Yes	Yes
newegg.com	Yes	-	Yes	No
wsj.com	No	No	No	-
expedia.com	Yes	-	Yes	Yes
theguardian.com	Yes	-	Yes	Yes
wunderground.com	Yes	-	Yes	Yes
southwest.com	Yes	-	Yes	Yes
nordstrom.com	No	No	No	-
gap.com	No	No	No	-
costco.com	Yes	-	Yes	Yes
ancestry.com	Yes	-	Yes	Yes
staples.com	No	No	No	-
npr.org	Yes	-	Yes	Yes
accuweather.com	Yes	-	Yes	Yes
time.com	No	No	No	-
constantcontact.com	Yes	-	Yes	Yes
okcupid.com	Yes	-	Yes	Yes
go.com	Yes	-	Yes	Yes
imdb.com	Yes	-	Yes	Yes
nytimes.com	Yes	-	Yes	Yes
walmart.com	Yes	-	Yes	Yes
weather.com	Yes	-	Yes	No
yelp.com	Yes	-	Yes	Yes
zillow.com	No	No	No	-
wordpress.com	Yes	-	Yes	Yes
groupon.com	Yes	-	Yes	Yes
cnet.com	No	No	No	-
stackoverflow.com	Yes	-	Yes	Yes
adobe.com	Yes	-	Yes	No
bleacherreport.com	Yes	-	Yes	Yes
twitch.tv	Yes	-	Yes	Yes
goodreads.com	Yes	-	Yes	Yes
tmz.com	Yes	-	Yes	Yes
woot.com	Yes	-	Yes	Yes
cbssports.com	Yes	-	Yes	Yes
engadget.com	Yes	-	Yes	Yes
surveymonkey.com	Yes	-	Yes	Yes
allrecipes.com	Yes	-	Yes	Yes
godaddy.com	No	Yes	Yes	Yes
nydailynews.com	Yes	-	Yes	Yes
united.com	Yes	-	Yes	Yes
kickstarter.com	Yes	-	Yes	Yes
aa.com	Yes	-	Yes	Yes
weebly.com	Yes	-	Yes	Yes
ticketmaster.com	Yes	-	Yes	Yes

coupons.com	No	No	No	-
jcpenny.com	No	No	No	-
houzz.com	No	No	No	-
zappos.com	No	Yes	Yes	Yes
delta.com	No	Yes	Yes	Yes
theverge.com	Yes	-	Yes	Yes
myfitnesspal.com	Yes	-	Yes	Yes
mailchimp.com	Yes	-	Yes	Yes
dell.com	Yes	-	Yes	No
wayfair.com	Yes	-	Yes	Yes
airbnb.com	Yes	-	Yes	Yes
hp.com	Yes	-	Yes	No
fitbit.com	No	No	No	-
whitepages.com	Yes	-	Yes	Yes
sbnation.com	Yes	-	Yes	Yes
t-mobile.com	Yes	-	Yes	No
theblaze.com	No	Yes	Yes	Yes
mint.com	Yes	-	Yes	Yes
steamcommunity.com	Yes	-	Yes	Yes
creditkarma.com	Yes	-	Yes	Yes
dailymotion.com	No	No	No	-
telegraph.co.uk	Yes	-	Yes	Yes
hootsuite.com	Yes	-	Yes	No
monster.com	Yes	-	Yes	Yes
evernote.com	No	No	No	-
alibaba.com	No	Yes	Yes	Yes
match.com	Yes	-	Yes	Yes
gamefaqs.com	Yes	-	Yes	Yes
rottentomatoes.com	Yes	-	Yes	Yes
marketwatch.com	Yes	-	Yes	Yes
bedbathandbeyond.com	No	No	No	-
qvc.com	No	Yes	Yes	Yes
nypost.com	No	Yes	Yes	Yes
hotels.com	No	No	No	-
fandango.com	No	Yes	Yes	Yes
barnesandnoble.com	Yes	-	Yes	Yes
bhphotovideo.com	No	Yes	Yes	Yes
photobucket.com	Yes	-	Yes	Yes
ebates.com	Yes	-	Yes	Yes
wix.com	No	No	No	-
webex.com	No	Yes	Yes	Yes
shutterfly.com	Yes	-	Yes	Yes
hilton.com	Yes	-	Yes	Yes
kbb.com	No	No	No	-
babycenter.com	Yes	-	Yes	Yes
instructables.com	Yes	-	Yes	Yes
addthis.com	Yes	-	Yes	Yes
barclaycardus.com	Yes	-	Yes	Yes
adcash.com	Yes	-	Yes	Yes
slideshare.net	Yes	-	Yes	Yes
forever21.com	Yes	-	Yes	Yes
evite.com	Yes	-	Yes	Yes
topix.com	No	No	No	-
bodybuilding.com	Yes	-	Yes	No
reuters.com	No	No	No	-
careerbuilder.com	Yes	-	Yes	Yes
archive.org	Yes	-	Yes	Yes
opentable.com	Yes	-	Yes	Yes
myway.com	Yes	-	Yes	Yes
tvguide.com	Yes	-	Yes	Yes
bankrate.com	No	Yes	Yes	Yes
zendesk.com	Yes	-	Yes	Yes
qualtrics.com	Yes	-	Yes	Yes
sourceforge.com	Yes	-	Yes	Yes

fool.com	No	No	No	-
sharesale.com	Yes	-	Yes	Yes
cars.com	No	Yes	Yes	Yes
biblegateway.com	Yes	-	Yes	Yes
bizjournals.com	No	No	No	-
cafemom.com	Yes	-	Yes	Yes
starbucks.com	Yes	-	Yes	Yes
thefreedictionary.com	No	Yes	Yes	Yes
refinery29.com	No	No	No	-
primewire.ag	Yes	-	Yes	Yes
stumbleupon.com	Yes	-	Yes	Yes
livestrong.com	Yes	-	Yes	Yes
officedepot.com	No	No	No	-
vox.com	Yes	-	Yes	Yes
battle.net	Yes	-	Yes	Yes
orbitz.com	No	Yes	Yes	Yes
rei.com	No	No	No	-
shopify.com	Yes	-	Yes	Yes
pogo.com	No	Yes	Yes	No
drugs.com	Yes	-	Yes	Yes
gamestop.com	Yes	-	Yes	Yes
xda-developers.com	Yes	-	Yes	Yes
thechive.com	No	Yes	Yes	Yes
kmart.com	No	No	No	-
aarp.org	No	Yes	Yes	Yes
shutterstock.com	Yes	-	Yes	Yes
angieslist.com	No	Yes	Yes	Yes
livejournal.com	Yes	-	Yes	Yes
tigerdirect.com	No	Yes	Yes	Yes
samsung.com	Yes	-	Yes	Yes
zergnet.com	Yes	-	Yes	Yes
stubhub.com	Yes	-	Yes	Yes
tomshardware.com	No	No	No	-
popads.net	Yes	-	Yes	Yes
ksl.com	Yes	-	Yes	Yes
seekingalpha.com	Yes	-	Yes	Yes
foxsports.com	No	No	No	-
speedtest.net	Yes	-	Yes	Yes
fanfiction.net	Yes	-	Yes	Yes
investopedia.com	Yes	-	Yes	Yes
macrumors.com	No	No	No	-
gamespot.com	No	Yes	Yes	Yes
disqus.com	Yes	-	Yes	Yes
wnd.com	No	No	No	-
fatwallet.com	Yes	-	Yes	Yes
basecamp.com	No	No	No	-
jcrew.com	No	Yes	Yes	Yes
arstechnica.com	No	No	No	-
liveleak.com	No	Yes	Yes	Yes
cisco.com	Yes	-	Yes	Yes
cc.com	No	No	No	-
victoriassecret.com	No	No	No	-
uproxx.com	No	No	No	-
espncricinfo.com	Yes	-	Yes	Yes
timeanddate.com	Yes	-	Yes	Yes
businessinsider.com	Yes	-	Yes	Yes
zulily.com	No	No	No	-
meetup.com	Yes	-	Yes	Yes
eventbrite.com	Yes	-	Yes	Yes
nfl.com	No	Yes	Yes	Yes
lifebuzz.com	Yes	-	Yes	Yes
deviantart.com	Yes	-	Yes	Yes
dailymail.co.uk	No	No	No	-
ign.com	Yes	-	Yes	Yes

retailmenot.com	Yes	-	Yes	Yes
mapquest.com	Yes	-	Yes	Yes
yellowpages.com	Yes	-	Yes	Yes
cnbc.com	No	No	No	-
priceline.com	Yes	-	Yes	Yes
change.org	No	No	No	-
city-data.com	Yes	-	Yes	Yes
signupgenius.com	Yes	-	Yes	Yes
howtogeek.com	No	No	No	-

Table D.2.: Detailed evaluation results of the CC.