CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Large-scale Detection of Cookie Paywalls

Adam Thunberg & Oskar Wallgren

Master's thesis in Computer science and engineering

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Large-scale Detection of Cookie Paywalls

Supervisor: Victor Morel, Computer Science and Engineering
Examiner: Vincenzo Gulisano, Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Abstract

Targeted advertising has become standard practice for websites that offer free content, at the expense of tracking visitors and selling their data to advertisers. There are alternative practices, such as funding from paywalls that block content until a fee has been paid. A new type of paywall was recognized in 2022, coined a **cookie paywall**, which differs from a regular paywall by offering *two* alternatives for access to content: agree to be tracked by consenting to cookies, or pay a fee. The ePrivacy Directive, by the European Union, states that: "A user cannot be denied access to a website for the purpose of declining cookies", a point of which cookie paywalls are clearly in violation. Despite this fact, they have been stated as allowed, or as a legal gray area in countries such as Austria and France, respectively. In the only known previous study on cookie paywalls, 2800 websites based in Central Europe were manually browsed and analyzed, where 13 websites were reported to be employing cookie paywalls. Our goal for this thesis was to study the prevalence of cookie paywalls on a large scale. We built a scalable web crawler that performed cookie paywall detection on the top 1 million most popular websites. We found 431 cookie paywalls, most of which were located in Germany, a region where cookie paywalls are prohibited by law. The runners-up were France and Italy, where 42 and 27 cookie paywalls were found, respectively. Only 8 were found outside of Europe. The price across all known cookie paywalls averaged €3.34 per month, ranging from €0.75 to €49 per month. In scalability tests, the crawler cluster achieved a linear increase in performance as more crawlers were added, reaching a throughput of 7.46 *pages/second* with 32 crawlers. Although the Crawler found hundreds of cookie paywalls, its detection algorithm was likely biased as it was built by analyzing websites only in Central Europe, leaving room for improvement. It is also possible to determine a trend of cookie paywall prevalence on the Web by performing similar analyses in the future.

Keywords: cookies, crawling, paywalls, web privacy

Acknowledgments

We want to thank Victor Morel, our supervisor, for conceiving the idea for this project, supporting us along the way, and for the cookies he baked. We also want to thank Vincenzo Gulisano for examining this thesis and keeping us grounded.

Adam Thunberg & Oskar Wallgren, Gothenburg, 2023-07-05

Contents

Chapter 1

Introduction

Targeted advertising is a common business practice for web services that offer free content. However, this monetary model comes at the expense of providing advertisers with insight into users' behaviors and interests. In 2018, the European Union passed the General Data Protection Regulation (GDPR). This change and the EU's ePrivacy Directive set requirements regarding the elicitation of consent for tracking users with web cookies [1]. The intent of these efforts by the EU was to give internet users more control over their personal information.

To address compliance with these laws, websites needed new ways of eliciting user consent. An increasingly common method has been through cookie banners [2] (see Figure 1.1). They were present before the GDPR, but have been developed since to address the new requirements. However, a study by Matte *et al.* [3] found that some websites might still track you even when rejecting cookies.

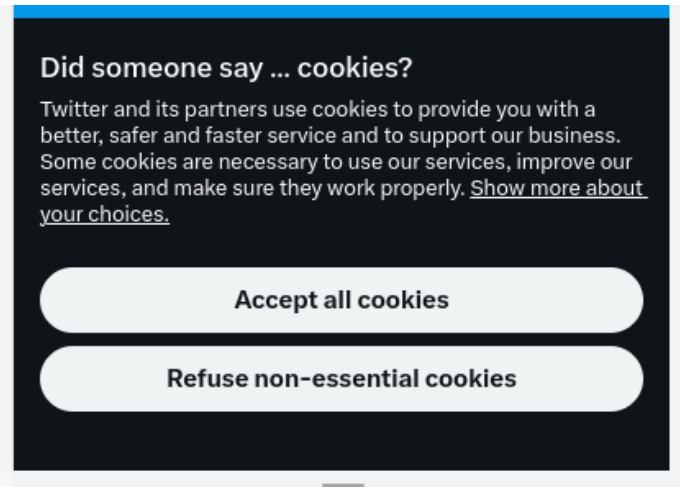


Figure 1.1: A typical cookie banner.

Some web services also seek to subtly manipulate the user into accepting cookies. Such practice is known as *deceptive design* [4] or *dark patterns*. An example of this is when a web page has a "reject cookies" button that is "considerably" less visible than the corresponding "accept cookies" button [5]. Such practice is not in

accordance with the GDPR, and additional examples have been found by the EU Cookie Taskforce [6].

1.1 Cookie Paywalls

The growing awareness and discourse about online integrity among Internet users [7], [8], along with the aforementioned laws, is a possible explanation for the rise of a recently identified strategy for web services: **cookie paywalls** [9]. A regular paywall typically prevents website content access unless a certain amount of money is paid. A cookie paywall is similar, as it also offers access in exchange for payment. The difference being an alternative option for site access in exchange for consent. A real example of such a paywall can be seen in Figure 1.2.

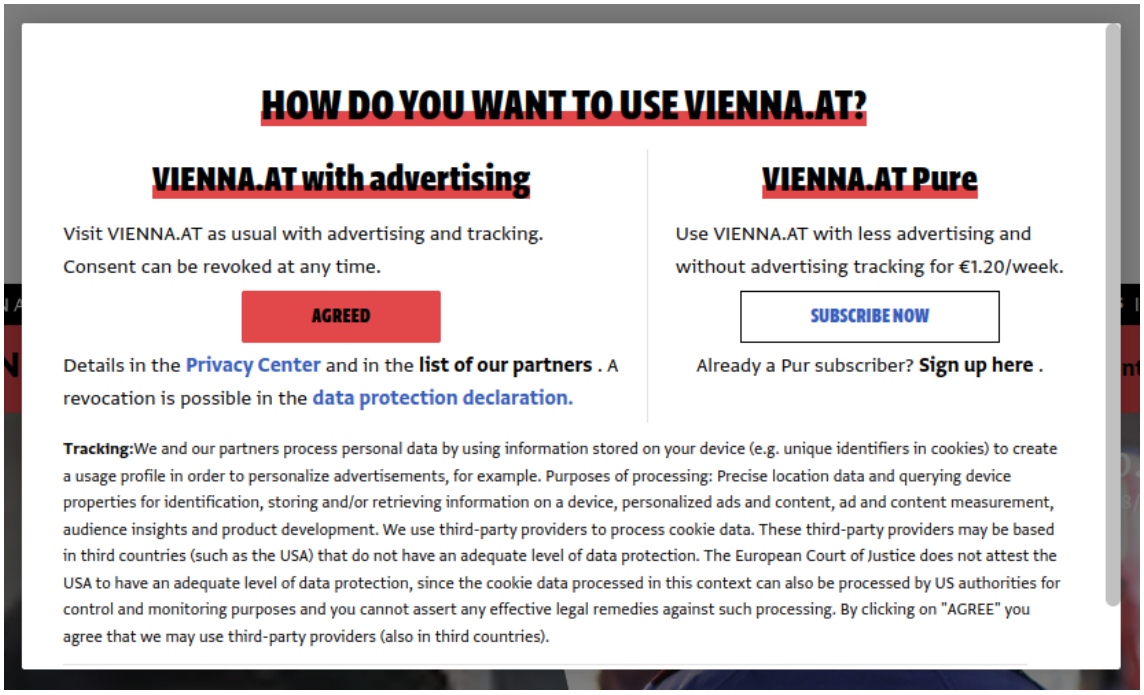


Figure 1.2: A Cookie Paywall found at *https://www.vienna.at* – Note the description of the right column: "Use VIENNA.AT ... without advertising tracking for €1.20/week", characterizing this type of paywall.

The rise of cookie paywalls is likely a consequence of website operators' financial dependence on selling user data. If users become less willing to consent to be tracked online as they become more aware of how their data is auctioned [10], it will be harder for websites to profit from targeted advertising. Not to mention online newspapers and similar industries in need of new subscription incentives [9].

In an effort to research cookie paywalls, Morel *et al.* [9] studied some of the most popular websites in Central Europe and found *13* websites that use paywalls of this type. Although the practice does not appear to be widespread, there is not yet enough data to make an accurate claim. The study, despite its small scale

and manual method, provides useful data to understand how cookie paywalls are presented to users.

1.2 Objective & Research Questions

Our goal is to investigate the prevalence of cookie paywalls on the Web by automating their detection on a large scale. Large-scale detection of cookie paywalls requires a tool that can scan large portions of the Web. Such a tool is commonly called a *web crawler*, which we have developed using a scalable computing approach. This approach, in theory, allows for an arbitrarily large crawl, only limited by computing resources. Important features of web pages have been extracted by the web crawler to detect cookie paywalls. The data has been gathered on a large scale and an analysis has been conducted to provide further insight into the prevalence of this type of paywall. The dataset contains information such as the website address, various extracted features, and the timestamp when it was crawled. Most importantly, it shows whether a cookie paywall was detected. In the case of paywall presence, data is gathered about paywall cost, placement, and payment model. The payment model could be a subscription or a one-time cost.

Our main focus lies in answering the following research questions:

RQ1 Where, geographically, are cookie paywalls most prevalent?

RQ2 Do local legal decisions concerning cookie paywalls have an apparent effect on the prevalence?

RQ3 Which types of websites (ex. newspapers, forums) typically employ cookie paywalls?

RQ4 What do cookie paywalls typically cost?

RQ5 Are payment models usually in the form of a subscription or a one-time cost?

RQ6 How is a scalable cookie paywall detector implemented?

RQ7 What are the main factors influencing the scalability of the Crawler?

1.3 Report Outline

This report proceeds with a technical background in Chapter 2, describing important concepts and technologies used in the project. Related work in Chapter 3 continues with discussions on previous work on cookie paywalls, natural language processing, and web crawling. Afterward, the design of the Crawler in Chapter 4 is detailed, including its scalable form and the cookie paywall detection algorithm. Furthermore, we show how this design is implemented in practice in Chapter 5. Tests and results are presented in Chapter 6, followed by a discussion and conclusion of these results in Chapter 7 and Chapter 8 respectively.

RQ1 and RQ2 are discussed in Section 7.1.1, where we consider the 431 confirmed cookie paywalls that we found during this thesis by crawling 1 million websites.

These cookie paywalls are listed in Table 4 in Appendix C.

RQ3 concerns the distribution of categories that sites with cookie paywalls belong to. For instance, Morel *et al.* [9] found that newspapers were the typical employers of cookie paywalls, but other categories were found as well, such as subject-specific magazines. Discussion concerning these results can be found in Section 7.1.2.

RQ4 was answered by running statistics like average, minimum, and maximum of prices observed from found cookie paywalls in Section 7.1.3.

RQ5 was answered by browsing each found cookie paywall website and recording their payment model. This was also discussed in Section 7.1.3.

RQ6 concerns the process of building our crawler, as detailed in Chapter 5. Scaling is achieved by using Container Orchestration Software Kubernetes, described in Section 5.2.

RQ7 was answered by analyzing how throughput, CPU and memory increased as the Crawler cluster grew in size, in Section 6.3.

Chapter 2

Background

This chapter describes terms and concepts concerning Web crawling, Web scraping, scalable computing, and Natural Language Processing (NLP) in the context of this project. This section begins with a legal background concerning cookie paywalls.

2.1 ePrivacy and GDPR

Web cookies (or just *cookies*) have been in use since the early 1990s, serving as a primary medium for persisting data between browsing sessions [11]. A common use is to keep a user logged in between HTTP requests. However, they can also be used to identify users across the Web as they browse, meaning they are a common method for user tracking. In the EU, the collection and processing of cookies are largely regulated by two legal texts: the ePrivacy Directive (ePD) for their collection, and the GDPR for their processing.

Since 2002, the ePD has required sites to ask for users' consent before collecting their data [1]. The GDPR defines consent in Article 4(11) as *"any freely given, specific, informed, and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her."* [12]. Throughout its 88 pages, the GDPR only mentions cookies once in recital 30, where it states that cookies are treated as personal data [1], [12]. The GDPR also states that the processing of personal data should be transparent and comprehensible to the user [11]. This means that the user should easily be able to understand what data will be collected by each cookie, and by whom.

Unlike the GDPR, the ePD includes more specific rules on cookie compliance, with the following key points [1]:

1. Aside from cookies strictly necessary for the operation of a web page, user consent must be obtained for the usage of cookies.

2. The purpose of each cookie must be clearly, visibly, and unambiguously explained.

3. A user cannot be denied access to a website for declining cookies.
4. User consent must be recorded.
5. Withdrawing consent must be as easy as initially giving it (also stated in the GDPR).

Cookie paywalls are seen as compliant in some regions, despite denying users access to site content [9]. In Austria, it is considered compliant because it gives the user the choice to be tracked or not. This differs from France, where each case should be assessed individually according to certain criteria [9]. It is worth noting that even though the GDPR and ePD are different legal texts, they overlap on several issues. Failure to comply with the GDPR or ePD can result in fines [11].

2.2 Web Page Technology

When loading a website with a web browser, there are several technologies and mechanisms in play. The process starts when a website address is entered in the address field of a browser, and a request is sent to a web server.

Web browsers and websites communicate using the Hypertext Transfer Protocol (HTTP). HTTP is a protocol for transmitting hypermedia documents, such as HTML, CSS, or JavaScript [13]. The protocol follows the client-server architecture, meaning a server waits for requests from clients and responds accordingly.

The response to the HTTP request contains a HyperText Markup Language (HTML) document; with HTML being the standard for structuring Web pages [14]. HTML introduces a tree-like structure in which each node is defined by a tag, as shown in Listing 2 in Appendix B. The root tag, `<html>`, has two children, `<head>` and `<body>`. The browser parses the HTML code and constructs a Document Object Model (DOM) tree that represents the structure of the web page.

In modern web browsers, it is possible to have more complex HTML structures with elements such as *iframes* and/or shadow DOM's. An iframe is a tag in the tree that represents a self-contained HTML document. This allows web pages to have nested pages, each with its own environment and session data.

A shadow DOM shares some functionality with iframes, such that it is an independent part of the DOM. The main purpose of having a shadow DOM is to have separate rules and styling for a part of the DOM, while also being more difficult to tamper with. The shadow DOM consists mainly of four elements: the shadow host, shadow tree, shadow boundary, and shadow root, illustrated in Figure 2.1. The shadow tree is not easily accessible from the main document, which protects it from unintended modification. Chromium-based browsers provide an API for access to these elements by calling the shadow root from the shadow host element [15].

During parsing, the browser loads eventual Cascading Styling Sheets (CSS), which specify the appearance and style of the HTML. While HTML defines the content of a web page, CSS defines how the content is presented through layout, typography, colors, and other visual aspects of the web page. To apply styling to an element, CSS

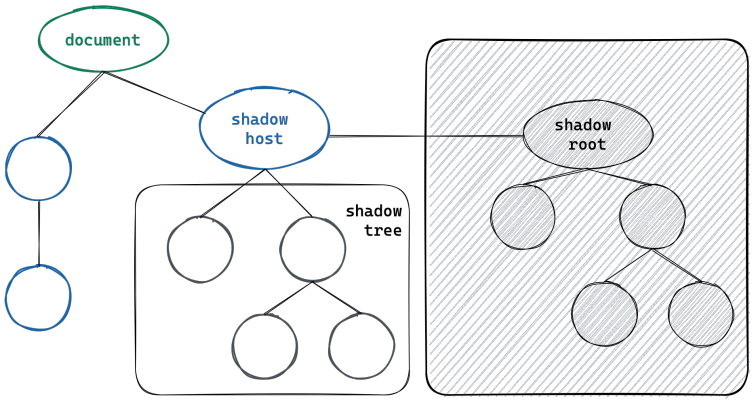


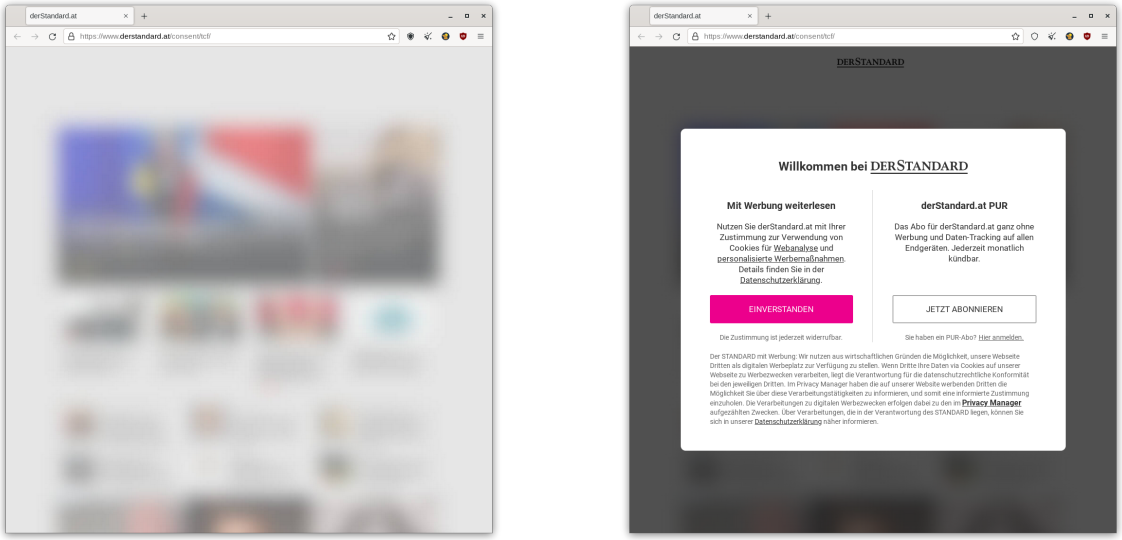
Figure 2.1: The shadow DOM tree structure

makes use of selectors that can query elements based on a tag name, tag attributes, and their properties. There are many styling properties with self-explanatory names such as *width*, *height*, *color*, and *font-family*. An important element positioning attribute is the z-index, which determines the stack order of visible elements. If two elements overlap in two-dimensional space, one with a higher z-index will obscure the other.

CSS can also determine whether an element is visible or not with the *display* property. Invisible elements can be seen in the HTML tree, but are not rendered by the browser. One potential reason for having non-visible elements is when visibility is determined dynamically based on user actions, credentials, or session details such as browser version. For example, if a user has an active subscription to a website, a cookie paywall should be hidden.

In addition to CSS, the HTML tree can contain JavaScript, which is a scripting language that enables dynamic behavior on Web pages [16]. HTML by itself is limited to static behavior, while JavaScript allows web pages to be modified without having to reload them. In the context of this project, it is most important to understand that JavaScript allows *asynchronous function calls* [17]. These types of function calls are non-blocking, meaning that they will not interrupt other parts of the program flow unless explicitly told to do so. They also have no *inherent* guarantee when they will terminate and with what result. This means that something like a cookie paywall may not be present immediately when the HTML document loads, if it is displayed as a result of an asynchronous function call. However, after an unspecified amount of time, or after a user action, an asynchronous function call may cause the element containing the cookie paywall to be loaded and added. Figure 2.2a shows what such a page may look like when fetching it with JavaScript disabled. Figure 2.2b shows the intended result with JavaScript enabled.

Figure 2.2 shows an example of how the rendering of a modal window depends on dynamic function calls in JavaScript. A **modal window**, or just **modal**, is a graphical window that takes precedence over content in the background. Precedence is used in how the user is meant to interact exclusively with the modal until it is closed.



(a) JavaScript disabled

(b) JavaScript enabled

Figure 2.2: This web page loads the consent form with a JavaScript asynchronous function call.

A common use for modals in the context of the Web is when a paywall is blocking access to the main content of a web page. The user is meant to resolve the paywall by paying a fee, after which the modal no longer blocks access for the user. Examples of modal windows that contain cookie paywalls can be seen in Figure 1.2 and Figure 2.2b.

In addition to the HTML in an HTTP response, the web server includes cookies intended to be stored in the user's browser. Cookies are small pieces of data that serve as one of the primary means of persisting data between web browsing sessions and can be used to store any form of data as long as it doesn't exceed memory limits [18], [19]. Examples of data that cookies are used for are:

- Session details like login status or user preferences
- Shopping cart data from an online store
- Tracking and Analytics
  - Previous visited pages
  - Time spent on a specific page
  - Actions such as clicked buttons and links

2.3 Web Crawling & Scraping

In order to perform a large-scale analysis on the Web, we have made use of web crawling. Web crawling is the process of systematically browsing a list of web pages [20]. Throughout this report, we will use the term Uniform Resource Locator

(URL) when referring to addresses pointing to websites.

Web *scraping* is the method of extracting and storing data from websites. In the context of this project, it can be defined as sending an HTTP request to a public host running a web server, and parsing and storing the HTTP response [21]. Parsing the HTTP response is important because it may contain extraneous elements of code and HTML syntax not needed for the purpose of the Crawlers analysis. In our case, we are primarily interested in plain text, meaning storage requirements are reduced as opposed to when storing entire HTML documents.

2.3.1 Selenium

Web crawlers do not necessarily always use real browsers when visiting websites, meaning crawled websites don't necessarily reflect the user experience. Selenium WebDriver addresses this issue by providing an interface that facilitates the automation of web browsers, enabling various operations involved in web crawling [22].

Selenium WebDriver is an open-source web browser testing framework. The framework's cross-browser compatibility allows developers to automate interactions with web pages using some of the most popular web browsers, including Google Chrome, Apple Safari, Microsoft Edge, Mozilla Firefox, and Internet Explorer [23]. By simulating user interactions, WebDriver can navigate through web pages by clicking links, submitting forms, and reading data from elements.

One particularly relevant capability of WebDriver, for this project, is its element locator functions. These functions can locate HTML elements by ID, class name, CSS selectors, and XPath.

To manage the synchronization between web crawling actions and element availability, WebDriver provides implicit and explicit waits. These wait mechanisms allow developers to define conditions that ensure the Crawler waits for elements to appear or generally meet any criteria before proceeding with data extraction. There are also timeout waits where an error is generated if a condition is not met within a certain time frame. WebDriver can also execute JavaScript code, and manage cookies and browser settings.

There are various configurations in which Selenium can run. One such configuration is the standalone WebDriver mode, which drives a browser natively and is spawned and accessed directly by a program. There is no inherent limitation in using this mode, however, it isn't designed for workloads at scale. It is primarily useful when wanting to run small-scale browser tests.

The other configuration is called Selenium Grid, which is designed to run tests in parallel against multiple (potentially different) browsers and even different operating systems. And importantly, it is designed to be scalable. As opposed to standalone WebDriver, Selenium Grid runs as its own server, which is accessible through a "hub". As explained in Selenium's official documentation [24], one can configure such a hub to which a client can send requests in order to run browser tests. However, the Hub is useless on its own and needs browsers to connect to it. The architecture is

visualized in Figure 2.3.

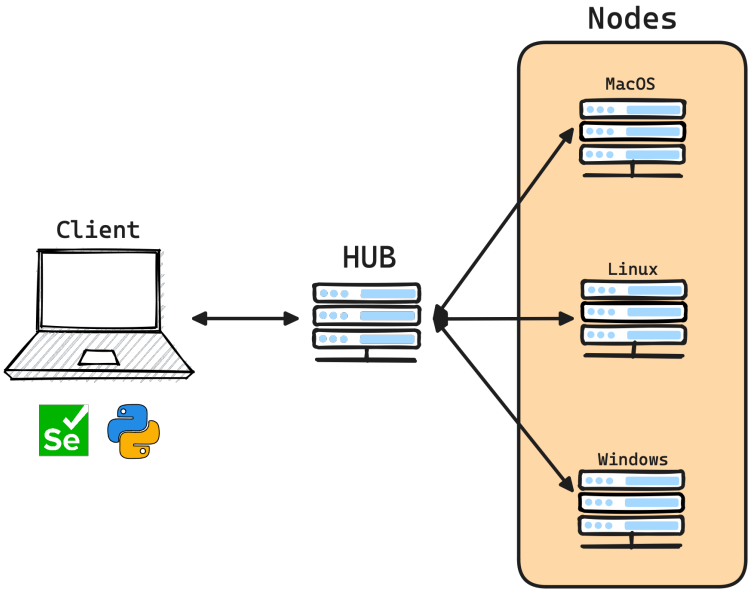


Figure 2.3: Selenium Grid Architecture

Any number of browsers can join the Hub, and each browser facilitates tests for one browser version. For instance, the Hub can be configured to have 8 Firefox browsers and 8 Chrome browsers attached to it. When a client connects to the Hub and requests sessions with one Firefox and one Chrome browser, the Hub will reserve one of each browser for this client.

2.4 Containers/Virtualization

There are numerous potential challenges when deploying many applications. For instance, multiple applications may require different versions of the same library to be installed; how should these libraries be installed and accessed? Additionally, how can such applications (along with dependencies) be installed on different computers, each with potentially different operating systems, while achieving the same behavior? To address these issues when building our crawler, container technology has been utilized [25]. Containers are software images that can be deployed repeatedly while retaining identical behavior. Containers are also *isolated* in terms of computing resources, meaning that errors and failures are less likely to affect other containers.

Containers "encapsulate" applications, along with configuration files, libraries and other dependencies, meaning they can be built and deployed to run with a certain set of libraries without conflicting with the libraries of other containers. Also, because containers run in reproducible container environments, they can be installed on any operating system that is capable of installing the *container runtime* in which the containers run.

Among the most popular solutions to build, manage and run containers is with Docker. Docker is a set of products but is mainly known for its container runtime.

Docker integrates tightly with Kubernetes (discussed in Section 2.4.1), as Kubernetes configurations, often by default, pull containers from Docker Hub. Docker Hub is a public container image registry, allowing anyone with internet access to create an account and develop and ship applications. However, systems like Kubernetes or Podman (explained below) can interface transparently with Docker images due to the standardized container runtime 'Containerd'.

Podman is an open-source Linux tool that also allows the building, management, and running of containers. As mentioned previously, it conforms to the 'Containerd' standard, which is part of the Open Containers Initiative [26]. Interestingly, it also works as a drop-in replacement for the Docker runtime, allowing users to run commands identical to when using Docker.

This project has made use of Podman when building, uploading, and running the Crawler container image, but commands are shown as if docker is being used. As mentioned, Podman works as a drop-in replacement and can be aliased in a bash shell environment with the following command: `alias docker="podman"`.

2.4.1 Container Orchestration Software with Kubernetes

Manually handling a cluster of numerous containers can be problematic, especially when having to repeatedly configure or restart them. Additionally, it may be challenging to manually keep track of and handle potential crashes and errors of multiple containers. Finally, there is the issue of deploying a cluster of containers in a reproducible manner when wanting to achieve highly specific application, network and computing configurations.

We have addressed these issues using *Container Orchestration Software* (COS). We made use of the COS called *Kubernetes* [27], which is portable and declarative. 'Portable' means one deployed Kubernetes configuration should behave the same on one machine as on another. 'Declarative' means the system's structure and behavior can be specified and applied entirely with configuration files. Kubernetes also has restart policies, which allows administrators to state what should happen in the case of errors.

There are multiple ways of deploying a Kubernetes cluster. One of the most convenient and simple methods is through Canonical's *microk8s*. It is installed through Ubuntu's package manager, *Snapd*. One of Snapd's main goals is to provide an interface through which packages can be installed on any device running Linux, regardless of which distribution or architecture is being used. Microk8s allows for a single-command installation of a Kubernetes cluster and has convenient methods in order to install important components like CoreDNS. Another way to deploy a cluster is by using managed Kubernetes services, like Amazon's Elastic Kubernetes Service (EKS), providing a similarly easy setup.

To facilitate a deeper understanding of the work in this report, it is important to be familiar with some terms associated with Kubernetes. The most important ones in this project are Nodes, clusters, containers, deployments, replicas, services,

namespaces, pods, and persistent volumes. A brief explanation of each of these concepts can be found below:

Nodes A node runs workloads. Workloads are applications running on Kubernetes, effectively meaning nodes are responsible for running applications. A node could be any virtual or physical machine capable of running Kubernetes.

Cluster A Kubernetes Cluster is a set of nodes running workloads.

Containers Container technology is explained in Section 2.4.

Replicas A replica is, in effect, a running container image. However, when using replicas in Kubernetes, it is a reference to how many copies (replicas) of a container are in question.

Namespaces A namespace isolates resources from other namespaces within a Kubernetes cluster. Namespaced objects can include deployments, pods, services, and more.

Pods A pod is the smallest, manageable unit of computing in Kubernetes, which groups one or more containers within an isolated environment. Containers within a pod behave as though they are all on the same machine, allowing them to communicate freely. They also share storage and network resources.

Deployments A deployment is a way of declaring a desired workload. For instance, a deployment can be used to define how many replicas of an application should run, what volumes to use, what computing resources it should have access to and more. Deployments are one of the most common ways of deploying applications in Kubernetes.

Services A service in Kubernetes allows the exposition of a container through a network. There is high customizability, such as only exposing within a namespace, or to the host machine.

PersistentVolumes A PersistentVolume (PV) is an either manually or dynamically provisioned storage unit, to be used for persisting data in workloads (like a deployment). A **PersistentVolumeClaim** (PVC) is a way to request and claim a piece of storage from a PV.

2.5 Natural Language Processing

Managing and analyzing large volumes of unstructured text data can be challenging due to the inherent complexity and variability of human language. These problems are addressed by the field of Natural Language Processing (NLP), which helps machines in understanding and extracting information from natural language data.

NLP typically begins by preparing text for feature extraction. This step is called text preprocessing, where a series of steps are performed to strip down and simplify the text and prepare it for analysis. Tokenization, stop word removal, and stemming are some common techniques for removing redundant information, as explained below.

Tokenization is the process of normalizing and breaking text into smaller pieces. Normalization refers to a collection of steps aimed at removing noise, such as misspellings and converting all letters to lowercase. Once the text has been normalized, it is broken down into individual units, or tokens. These tokens can be as small as a word or as large as a sentence, depending on the task. This process facilitates text analysis by transforming the unstructured data into a structured format.

Stop word removal refers to the elimination of common words within a text in a given language. Stop words are common to different texts, regardless of their subject, making their exclusion beneficial in the process of text categorization. These stop words, which are defined in a specific language corpus, can be conveniently accessed using the Natural Language Tool Kit (NLTK) [28], a well-known NLP library designed for Python.

Stemming is the process of reducing words to stems, meaning that different inflections of the same word ideally result in the same stem. An example is when a stemmer returns "Advertise" for both "Advertiser" and "Advertisement" as input. This can greatly simplify text analysis since different inflections of words can be treated equally. There are a number of different stemming methods, but we found the Porter Stemmer to be sufficient for our use case [29].

Bigrams, trigrams, and n-grams are a set of two, three, or any number of adjacent symbols in a piece of text. An example of a trigram is "Without advertising tracking", while an n-gram denotes a phrase with n tokens.

There are additional preprocessing techniques, such as part of speech tagging and named entity recognition, that further facilitate text analysis [30], [31].

After text preprocessing, feature extraction is performed to determine properties of a given text. These properties can in turn be used for categorization or sentiment analysis of the text. Examples of some features are the number of words, the frequency of certain words, or the average sentence length. Text properties can also be used to determine additional properties, such as whether the text describes a cookie paywall. Somewhat more complex features are the presence of certain n-grams. An example of such a feature is searching for the phrase: "subscribe without tracking", which is a trigram.

2.6 Consent Management Providers

Asking users for consent to collect and process user data, while complying with the ePD and GDPR, can be complicated. As a result, companies providing solutions called Consent Management Providers (CMPs) have emerged to assist other companies in managing consent. CMPs provide websites with services that claim to ensure consent is obtained before users' data is processed; such a service can be called *Consent as a Service* (CaaS) [32].

From the perspective of website users, CMPs often appear as cookie banners, consent modals, or consent pop-ups like the one in Figure 2.4. When the individual gives

consent, their data is often sent to an advertiser network that connects publishers and advertisers. Advertisers are entities that seek to market something to a specific group of users, and publishers are entities that sell ad space on their websites. The result is targeted advertising, where each user receives a unique collection of ads based on the data collected by a CMP.

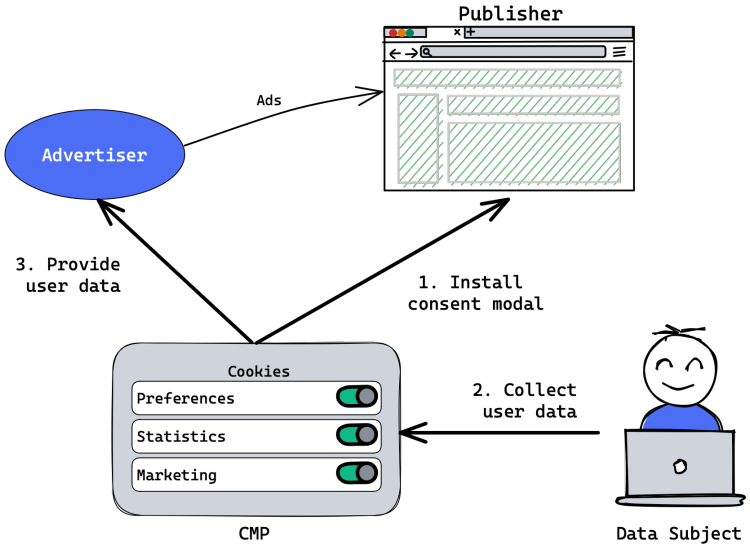


Figure 2.4: CMP information flow. 1. The publisher installs a CMP on their website to collect consent from users. 2. A data subject interacts with the CMP element, giving consent to collect their data. 3. Based on user data, an advertiser purchases ad space from the publisher.

Chapter 3

Related Work

This chapter is a summary of the state-of-the-art research related to this thesis. We cover three areas on which this research is based: cookie paywalls, language analysis, and scalable crawling.

3.1 Cookie Paywalls & Paywall Detection

To the best of our knowledge, the only previous study on cookie paywalls was conducted by Morel *et al.* [9], who also coined the term. The study was based on a manual classification of the most popular websites in 13 Central European countries. They only analyzed 2800 websites and found 13 websites employing cookie paywalls. Despite its manual approach and small scale, it provides a necessary starting point. They used a heuristic method based on features of the language of cookie paywalls to detect them. In addition to identifying sites with cookie paywalls, the legality of this practice in European countries was researched. Although the ePD states that a user cannot be denied access for refusing cookies, the authors found that some countries' data protection authorities (DPAs) consider it permissible. The Austrian DPA says that cookie paywalls constitute valid consent, while the French DPA requires that each case be evaluated individually. The authors also found that the German and Danish DPAs consider cookie paywalls to violate freely given consent [9]. The 13 cookie paywall websites were analyzed further to determine data such as paywall type, website category, and price/type of subscription. We have built upon the work of this study by performing similar analyses on a larger scale.

Papadopoulos *et al.* [33] attempted to automate the detection and classification of paywalls on the Web involving machine learning; however, the study does not address cookie paywalls. The authors did, however, conduct a thorough review of the types of sites that employ paywalls and their country of origin. Finally, they analyzed how paywalls affected user privacy.

Another similar paper studied cookie banners and CMPs to assess their compliance with GDPR [34]. They found that 94.7% of 29,398 had at least one issue concerning the GDPR, with issues including implicit consent, incorrect expiry of cookies, ignored

choices, and more.

3.2 Language Analysis

The analysis of website content, such as cookie paywalls, is largely based on language. As mentioned by Hashemi [35], Web page classification has proved essential for Web indexing and topic-specific research on the Internet. The study discusses numerous technical approaches, using both NLP and machine learning, in order to detect criminal activities on the Web, such as sex trafficking [36] and solicitation. Their approach is relevant given that our goal is to classify web pages as *with* or *without* a cookie paywall, using similar tools.

Hamdy [37] used pre-trained transformer-based models to make a binary classification of tweets. The classification determined whether a tweet had offensive language or not. Such practice bears many similarities to how the language of cookie paywalls can be classified. Both our and their method can be viewed as large-scale web crawlers that analyze language to determine a specific property of text on a web page.

Deng *et al.* [38] developed a tool to classify web pages based on HTML tree structure using multiple classifiers. A similar strategy could be tried for classifying sites as cookie paywall sites. However, there may not be any distinguishing pattern in how cookie paywall prompts are placed in such a structure.

3.3 Scalable Web Crawling

Web crawlers have a long history [39], with the earliest one being written in 1993 [40]. Among the most famous is "GoogleBot" [41], responsible for powering Google's search engine. GoogleBot indexes web pages in depth up to 15 MB per page. Because online services must obtain the user's consent before collecting their data, as discussed in Section 2.1, elements such as cookie banners tend to appear immediately to the user, limiting the analysis to the landing page.

We are building a scalable web crawling architecture, closely resembling one by Ansuman Prusty *et al.* [42]. They, too, use a 'master program' for delegating work while scaling horizontally by deploying new containers in a Kubernetes cluster. We have drawn inspiration from their approach and how they assessed their scalability. For caching job requests, they used Redis Caches, and for storing web documents, they used Object Storage in S3. This differs from our approach of simply using one central database for all crawler and delegator communication. While using an in-memory database such as Redis certainly improves performance, we did not consider it to be necessary for our crawler, as we did not perform deeper link navigation of websites.

Eyzenakh *et al.* [43] implemented a distributed crawling solution, based on the Python scraping framework Scrapy, running in a Kubernetes cluster. The proposed solution scales well, but unlike Selenium, does not use actual browsers when visiting

web pages. This means dynamic behavior as a result of JavaScript would not be caught.

An early comparison of parallel crawlers by Cho and Garcia-Molina [44] discusses the potential problem of "overlap", i.e., when more than one crawling process visits the same page; a problem addressed in Section 4.4.1. They also discuss "quality", referring to which URLs are crawled and in which order, which we discuss in Section 5.3. Our quality inherently differed, as they collected most of their URLs from links inside the web pages taken from an initial URL pool. Our crawler did not navigate deeper inside any website than the landing page, due to the nature of a cookie paywall (which should appear immediately when loading a page). Additionally, their crawler ran a bigger risk of overlapping, as different websites may share links, creating an additional need for coordination between parallel crawlers. However, as we sample URLs from a website rankings list, there is a risk that multiple URLs may redirect to the same website. We have not discussed this possibility in this report.

Chaulagain *et al.* [45] assessed scalability and other possibilities (cost-efficiency, performance, elasticity) when deploying web crawlers on AWS. Like us, they used Selenium to script real web browsers and found that they could achieve a cost-effective, high-performance, and elastic solution using AWS services. However, they did not automate browser interactions such as mouse movements, which may cause content to be visible that would not be otherwise. They also scraped entire web pages indiscriminately, instead of identifying elements that fit certain criteria, such as modal windows.

Chapter 4

Design

This chapter first describes the design and intended behavior of the Crawler, along with the detection algorithm, in detail. Finally, the Crawler's scalable cluster design is described.

4.1 Crawler Program

The goal of the Crawler is to programmatically visit a given list of URLs (as outlined in Section 4.2), extract information from the resulting web pages, and apply the detection algorithm (as detailed in Section 4.3). Since the focus of our research questions is on functionality rather than usability, it was designed as a command-line application.

A crucial property of the Crawler is that it must use real web browsers when loading web pages, in order to observe how webpages behave when humans visit them. This is called dynamic behavior, occurring as a result of JavaScript running.

4.1.1 Filtering Webpage Content

Looking at the sites found by Morel *et al.* [9], most of them place the cookie paywall inside a modal window (as described in Section 2.2). However, it is possible to redirect users to a web page containing only the cookie paywall without a modal. Unfortunately, early crawling tests showed that indiscriminate analysis of entire web pages tends to produce false positives. Examples can be sites which sell tools to prevent tracking, such as cookie-banner interaction extensions. Another example could be a newspaper with an article about cookies. Therefore, we filter webpage content to extract modal/obscuring windows, on which we exclusively attempt to detect cookie paywalls.

In order for sites to obtain user consent as early as possible, cookie paywalls and cookie banners are likely to be presented on the first page. By definition, cookie paywalls block access to site content by either obscuring or replacing it entirely. Unlike modals, banners are non-blocking, meaning they don't prevent the user from accessing the content, meaning cookie banners cannot be paywalls.

To extract windows that obscure the background of a web page, a number of properties are examined. Our algorithm relies primarily on tag attributes of known CMPs, such as IDs, and visible elements with the highest z-index. Elements with the highest z-index tend to obscure content in a web page, which is characteristic of modal windows, as described in Section 2.2. The pseudocode of the process is shown in Algorithm 1.

Algorithm 1 Algorithm for extracting elements

```
1: if HTML contains known CMP then
2:     Return known CMP modal
3: end if
4: if HTML contains element(s) with z-index then
5:     Find the element with the highest z-index
6:     Make sure the element is displayed
7:     Return z-index element
8: end if
```

4.2 URL Sampling & Quality

In order to find cookie paywalls, we need a list of websites in which the paywalls could be located. Due to project duration and resource constraints, the number of sites to be crawled was limited to 1 million. Because of this limitation, we decided to use a list whose websites are reported as the top most frequently visited websites. We consider these sites to, on average, have the most impact on web users. In this case, "impact" refers to how often a website is visited, on average.

It is important that such a list is resistant to manipulation, for example by means of artificial ranking boosts through the use of bots. We deem such resistance important because we want to crawl a list of websites that, when compared to alternatives, more accurately represents the most popular part of the Web. We consider this part of the Web to be more representative for a couple of reasons. Firstly, websites that are artificially and intentionally boosted toward the top of a list are not placed there because human visitors browse these websites in their daily life. This means they likely do not have the same impact on human visitors as other websites in the same list. The impact on humans is emphasized, since cookie paywalls are specifically designed for *human* interaction. Secondly, websites which are boosted to the top may be boosted for the purpose of spreading malware. We want to reduce the risk, although the risk may be low, of spreading such malware through the cookie paywall websites we have published.

4.3 Cookie Paywall Detection

This section provides a detailed description of the cookie paywall detection algorithm. To detect cookie paywalls, we first identify the necessary information from web pages. As mentioned in Section 2.5, detection is based on deriving meaning from text on

web pages. Therefore, words, phrases, and sentences that are characteristic of cookie paywalls are considered relevant data. It is crucial to ensure that these text patterns are absent when cookie paywalls are not present to avoid false positives.

Initially, we heuristically analyzed cookie paywall phrases from the known sites identified by Morel *et al.* [9]. This analysis helped us to identify some initial patterns, which are keywords or phrases that, when combined, can be used to detect cookie paywalls from text.

Given that cookie paywalls may be presented in many languages, we have simplified the analysis to a single language through the use of a translator. This translation would be done on modal/obscuring windows (see Section 4.1.1), after which text preprocessing and feature extraction would occur, described in the next sections. We chose the single targeted language to be English, mainly because NLP libraries that we used had first-hand support for English. In order to translate other languages to English, a translator usually needs to know which language it should translate from. This presents a different challenge, as we need to determine which language a given web page is primarily written in (addressed in Section 5.1.2).

4.3.1 Text Preprocessing

The process of feature extraction for cookie paywall detection presents several key challenges. First, the presence of uninformative tokens, such as stop words, adds noise to the data and complicates the extraction process. This is further complicated by the inherent complexity of textual data. A second challenge is the accurate identification of price tags within the text, given the wide variety of currencies, formats, and symbols in which they can be found.

To address these challenges, we have developed an approach that uses standard text preprocessing techniques. These include removing noisy tokens such as stop words, as described in Section 2.5, and applying stemming to simplify the text. Finally, bigrams and trigrams are generated to allow the analysis of phrases consisting of two to three words.

To overcome the challenge of identifying different price tags, we use regular expressions. These allow us to match and extract multiple variations of price tag formats. Once the price tags are identified, they are replaced with a standardized symbol so that they can be treated a universal token in the feature extraction process.

4.3.2 Feature Extraction

Once the text has been preprocessed, involving filtering and stemming, the next step in the analysis is feature extraction. In this project, feature extraction aims to identify and capture the key characteristics that are typical of cookie paywalls. These characteristics are explained as follows:

Associated Price Cookie paywalls usually present the user with a price. Detecting the presence of price-related information in the text is critical to detecting the presence of a cookie paywall.

Common recurring words Sites with cookie paywalls often include certain words or phrases that recur across different instances of the paywall. These recurring words serve as additional indicators of the presence of a cookie paywall and help identify its characteristic patterns within the text.

Subscription-based model All currently known cookie paywalls offer a subscription-based payment model. This may distinguish cookie paywalls from other forms of content restriction, such as one-time purchases or ad-supported access.

Instant display and content blocking A typical property of cookie paywalls is their immediate presentation to users on a website. Unlike other types of content restrictions that may appear after some engagement or browsing, cookie paywalls aim to block access to content immediately. Identifying the cues that indicate this immediate presentation and subsequent content blocking may facilitate accurate detection.

4.3.3 Detection Algorithm

The detection algorithm's purpose is to use the features extracted to determine whether a website is likely to employ a cookie paywall or not. Deciding which features are the most important is not entirely simple. A combination of existing features could indicate that a cookie paywall is likely present, but making criteria too specific could result in many false negatives. On the other hand, making the criteria too general can result in many false positives. Because little research has been done so far on cookie paywalls, there is no known dataset (aside from our own) against which we could benchmark detection accuracy.

To address this challenge, we use the dataset of cookie paywalls found in the early stages of our research as a test dataset. The detection algorithm is tested and fine-tuned on this dataset to measure its accuracy in finding true positives. In combination with testing on this dataset, a heuristic approach is used to map phrases that are unique enough to avoid large amounts of false positives. Avoiding large numbers of false positives is essential to allow manual verification of each positive result within the project timeline.

Algorithm 2 outlines the process involved in cookie paywall detection:

4.4 Scalable Cluster Architecture

As we want to multiply the Crawler's performance by replicating (scaling) it, we need to adapt the program to work in a cluster of other crawlers, while avoiding bottlenecks. This adaption primarily concerns how the program will receive work and where results should be transmitted. In the context of this scalable architecture, the crawlers will be referred to using indefinite articles (i.e., *crawler*, or *crawlers*). The cluster, as a whole, will be referred to as *the* Crawler.

For the Crawler to be considered scalable, both performance (pages/second) and computing resource demands (in terms of CPU and memory) must grow linearly

Algorithm 2 Outline of the cookie paywall detection algorithm in the context of the crawling process

```
Receive body of website text
Tokenize text
if Text has a price tag then
    Replace price tag with fixed token
end if
Remove stop words from the text
Use PorterStemmer on text
Determine and store what words of interest are present
Determine and store what phrases of interest are present
if Text contains two words of interest and one n-gram of interest then
    Return LIKELY
else
    Return UNLIKELY
end if
```

with the number of crawlers in the cluster. This is because the scalability of a system is determined by its ability to efficiently handle increasing workloads.

The orchestration of crawlers is handled by Container Orchestration Software (discussed in Section 2.4.1). The role of COS is to start, configure, and recover crawlers as needed. There is also a program called the delegator which is designed to delegate URLs to crawlers, hence the name. It is also used to print various data from the database when requested. Figure 4.1 shows an abstraction of the intended configuration of the scalable crawler in terms of how data is communicated; and there is no temporality, i.e., everything can happen in parallel.

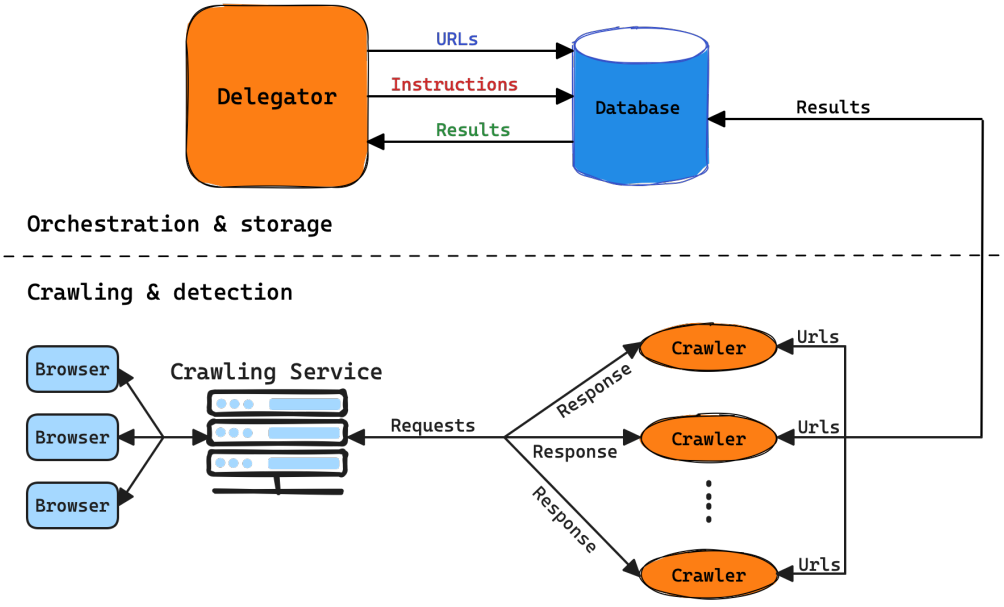


Figure 4.1: An abstracted view of our intended scalable configuration

To build the configuration from Figure 4.1 and achieve scalability, a number of requirements need to be met:

Avoiding redundant work Once a crawler retrieves any URL for crawling, it should not be retrieved by another crawler, unless the former crashes without producing a result for that URL. URLs that have been claimed but not completed should be considered unclaimed after a certain amount of time. Unclaimed URLs should be processed eventually.

This requirement is addressed by the URL selection algorithm described in Section 4.4.1.

Crawler restoration and unclaimed URLs Crawlers that crash should eventually be restored.

This requirement is addressed through container technology implemented in Section 5.2.1.

Simple cluster configuration Configuring the cluster, adding, and removing crawlers should be a straightforward task. This process should be achieved by modifying and applying a configuration file.

These requirements are addressed by the implementation of orchestration software implemented in Section 5.2.2.

Linear performance increase When a crawler is added to the cluster, the performance increase in the cluster should be at least linear. Performance refers to throughput in terms of how many pages the Crawler is capable of processing per second.

These properties are tested in Section 6.3.

4.4.1 URL Selection Algorithm

In order for multiple crawlers to run in a cluster, they must agree on which URLs to crawl, and in what order. This is where "URL overlap" comes into question, as mentioned in Section 3.3. The crawler program has been adapted to run in a cluster so that crawlers fetch URLs from an external database, as shown in Figure 4.1. A system is needed to ensure that different crawlers do not redundantly fetch the same URLs, while still crawling URLs in order. Primarily, the database table responsible for storing URLs and their results has the schematic as shown in Table 3. But, the most important fields to know of in this section are:

url Self-explanatory.

date_taken_<browser> When the URL was claimed.

date_finished_<browser> When a result was reported.

id_<browser> The ID of the claimant.

result_<browser> The result reported (error, likely or unlikely).

There is also the "registry" table, which contains three fields:

id Unique crawler identifier.

browser Browser utilized by the crawler.

last_report Timestamp of crawler's last interaction with the database.

The purpose of the registry is for the delegator to keep track of active crawlers. If a crawler crashes, its entry is removed after a fixed timeout, and associated URLs are cleared so that they can be claimed by a running crawler. In order for URL selection to work properly, we have utilized two algorithms: one for the delegator and a shared one for crawlers. The delegator's algorithm is presented in pseudocode in Algorithm 3.

Algorithm 3 Delegator Algorithm

```
1:  TIMEOUT ← 100          ▷ Seconds of inactivity before crawler is removed from registry
2:  N ← 4                  ▷ Max number of URLs to be assigned to a crawler
3:  DELAY ← 0.1            ▷ Seconds of wait after each loop
4:  while do
5:      Delete registry entries where last_report has exceeded TIMEOUT
6:      Declaim URLs claimed by inactive crawlers
7:      for all crawler ∈ registry do
8:          Select max. N URLs that are not finished and are claimed by crawler
9:          N_r ← number of selected URLs
10:         N_s ← N − N_r
11:         if N_s > 0 then
12:             Assign max. N_s unclaimed and unfinished URLs to crawler
13:         end if
14:     end for
15:     Wait for DELAY seconds
16: end while
```

The delegator runs Algorithm 3, which assigns unprocessed URLs to crawlers in the registry. This algorithm also clears the registry of inactive (i.e., crashed or unresponsive) crawlers. Claimed and unprocessed URLs claimed by inactive crawlers are eventually declaimed, allowing these URLs to be claimed and processed by an active crawler. N, the maximum amount of URLs to be assigned to any one crawler, was intentionally made small. A small N ensures fine granularity, so that a large portion of URLs in the final stage of a crawl are likely not assigned to fewer crawlers than there are available. Also, a portion of more than 4 URLs will not remain claimed by inactive crawlers, which is a tiny portion in the context of a million pages. The loop on line 4 repeats as long as the delegator is running. Line 5 deletes all inactive crawlers from the registry. Line 6 takes all URLs belonging to inactive crawlers and makes them available for claiming. The loop on line 7 iterates every *active* crawler. This loop assigns URLs for every active crawler such that all of them will have N unfinished URLs assigned to them, assuming there are enough available

URLs. When there are not enough available URLs left, some crawlers may process the remaining URLs while the others remain idle. This final phase of the crawl will involve a maximum of N URLs per crawler.

The process of selecting URLs for each crawler is shown in Algorithm 4. The function 'ping()' updates the registry to change 'last_report' to the present timestamp.

Algorithm 4 URL selection algorithm

1: *ping*()
2: Select all URLs assigned to this crawler by the delegator
3: Return selected list of URLs

This algorithm is run only when a crawler has finished working through its previously claimed URLs. In essence, the delegator's algorithm centralizes the claiming of URLs, meaning there will be no conflicts, which allows crawlers to fetch their assigned URLs without additional coordinating logic.

Chapter 5

Implementation

In this chapter, we detail the practical implementation of our scalable crawler. The details of this chapter include the specifics of the Crawler's mechanisms, the technological choices that underpinned its functionality, and the challenges encountered and overcome during its development.

5.1 Crawler Program

This section details how Selenium and BeautifulSoup were used to crawl, parse, and extract text from web pages. A flowchart of the steps the crawler takes to process a URL is visualized in Figure 5.1.

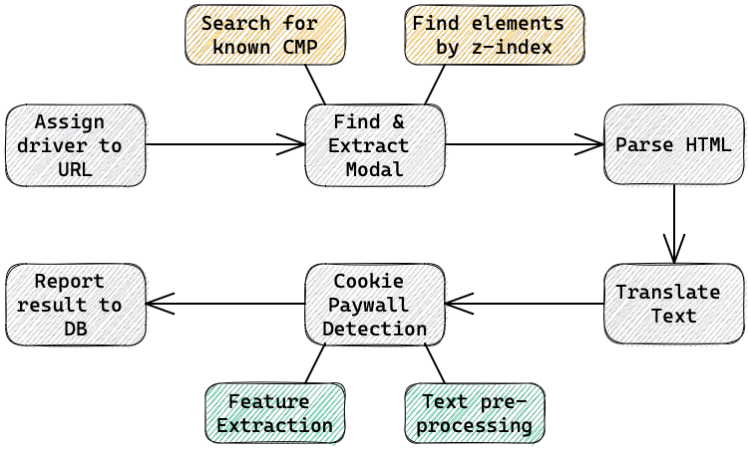


Figure 5.1: Flowchart of how the Crawler processes one website

All functionality of the crawler program has been developed using the programming language Python, which is a modern, object-oriented, imperative, and interpreted programming language [46]. It is commonly described as especially readable and easy to use due to its intuitive syntax [47]. It is also popular [48] and has numerous resources built for it. Python was chosen for the aforementioned reasons and because it is well-documented and has a variety of libraries related to web crawling & scraping.

The crawl begins with an initialization process that consists of two steps. First, it parses the command-line arguments that set the rules for how the crawl will run. Then it starts a crawl with the given configuration. The crawler program provides flexibility in starting and customizing a crawl by using different command line arguments, which are parsed using Python's built-in argument parser library *argparse*. A comprehensive list of available arguments and their descriptions can be found in Table 2, where each argument is described.

The Selenium library was used to run automated web browser tests, using actual web browsers. The crawler requests the rendered web page from loading a URL, and the page is used to query specific elements of the source according to certain criteria. Elements can be queried based on tag names or properties such as class or ID using CSS selectors [49]. An example of a command using Firefox as the driver, reading URLs from a file named `urls.json`, and with modal search enabled looks like this:

```
$ python src/crawl.py --firefox --file urls.json --modals
```

Once the initialization process is completed, the Crawler processes each URL in what is called the scraping process.

5.1.1 Extracting Modals

It is important to note that a large portion of any website may not be related to cookie paywalls. Therefore, analyzing a site in its entirety could result in a high rate of false positives. An example of misidentification can occur when analyzing an online newspaper. Such a site may have articles about Internet tracking that contain language similar to that found in cookie paywalls, resulting in a false positive. This highlights the usefulness of filtering website content, more specifically, extracting modals (described in Section 4.1.1).

To address this issue, we attempted to reliably extract modal windows from websites. Our modal extraction is based on two main factors: element IDs of known CMPs and the CSS property 'z-index'. An element from one CMP will usually have the same element ID across different websites, making it extractable using the Selenium Driver Interface. Selecting an element with the ID "cmp" with Selenium looks like this:

```
driver.find_element(By.ID, "cmp")
```

A list of recurring IDs that have been identified can be seen below:

- "cmpwrapper"
- "iubenda"
- "sp-message-container"
- "onetrust-banner-sdk"
- "cookie-wall"
- "cmp-modal"

- "didomi"

In addition to these identifiers, the algorithm looks for the HTML tag `<dialog>`, which is often used to create modals because it has built-in functionality to show, hide, and obscure content. The known CMPs were found by manually noting element IDs on sites found by an early, non-scalable version of the Crawler. However, it should be noted that CMPs may provide both cookie paywalls and/or cookie banners. This means that finding a known CMP is no guarantee that a cookie paywall is being used. This makes the detection algorithm equally important regardless of how the modal is found.

If no known CMP can be found, the algorithm extracts the element with the highest z-index. This element is usually a `<div>` acting as a wrapper for a modal. Getting the z-index of an element is possible in Selenium through the function call:

```
value_of_css_property("z-index")
```

There are a number of challenges to find the correct element using the z-index:

Non-displayed elements can be cookie banners or paywalls that have a z-index but are not visible. Since non-visible elements shouldn't be parsed, they are filtered out using the built-in Selenium function `element.is_visible()`.

Blurred Backgrounds are elements that draw attention away from the page's main content and instead focus on a modal window. One way to achieve this effect is by placing the background behind the modal using a lower z-index. However, in cases where the background has the same z-index as the modal, the algorithm may return an empty element. To avoid this issue, the algorithm first checks whether the found element contains any child nodes with text. If none are found, the algorithm proceeds to check the element with the second-highest z-index.

Shadow DOM, described in Section 2.2, hides entire trees in the DOM, making them invisible unless explicitly invoked. A cookie paywall inside a shadow tree will not be found by primitive web scraping because it is not present in static HTML. When querying a shadow element using one of the Selenium locators, such as `driver.find_element(By.ID, 'cmpbox')`, it will return a `NoSuchElementException`. To find cookie paywalls in the shadow DOM, it is required to search for the shadow host, which in turn has a reference to the shadow root. Figure 5.2 shows an example of how to get the shadow content on the website `https://hausgarten.net`.

```
shadow_host = driver.find_element(By.ID, 'cmpwrapper')
shadow_root = shadow_host.shadow_root
shadow_content = root.find_element(By.ID, 'cmpbox')
shadow_html = shadow_content.get_attribute('innerHTML')
# parse shadow_html with BS4 and extract text
```

The interface with the shadow DOM can be different for different browsers. The example in Figure 5.2 was done with a Chromium-based browser but would not work in for example Firefox. The main difference in Firefox is that `shadow_host` doesn't

Figure 5.2: Shadow DOM HTML example from `https://hausgarten.net`

have a reference to `shadow_root`. Instead, the shadow DOM is opened by injecting a JavaScript script into the browser. Once the shadow DOM is open, it is accessible using Selenium locators.

Other examples of HTML structures that can affect dynamic content include event listeners and window detectors. Event listeners can dynamically change HTML based on user activity, allowing asynchronous functions to be triggered only when user interaction is detected. There are sites that do not render cookie paywalls unless a user interacts with the site by moving the mouse, such as `https://www.auszeit.bio` and `https://www.derwesten.de`. This issue was addressed by automating mouse movements using Selenium's 'ActionChains'.

5.1.2 Parsing and Translation

Once a modal has been found, its content must be organized before cookie paywall detection can be applied. Effective feature extraction from multilingual web content requires a two-step approach: HTML parsing and then translation.

Initially, website data, inherently structured in HTML, is parsed down to plain text using the Python library BeautifulSoup [50]. The parsing process requires a non-trivial approach to text segmentation, as sentences within HTML often extend beyond individual tags. To overcome this challenge, the HTML data is stripped of all tags, preserving only tabs and whitespace. The resulting text is assembled into a list, purged of empty entries, and then combined into a single text string, thereby ensuring the integrity of the sentences.

Once the HTML has been converted to plain text, the text of non-English web pages is translated into English. This translation phase is essential for multilingual web crawling, as it allows NLP to focus on a single language. Focusing NLP on one language - in this case, English - streamlines processing and improves the overall efficiency of data extraction.

The translation is facilitated by the integration of AWS's Translate service into the crawler, enabling real-time translation. For optimization and speed, the source language is typically identified using the lang attribute embedded in the `<html>` or `<head>` tag. These tags usually contain a two-letter language code indicating the source language. In the absence of such tags, an 'auto' value is assigned to the translator, which then uses AWS Comprehend, a machine learning service, to determine the source language. Using the 'auto' value incurs additional costs within

AWS.

5.1.3 Detection Algorithm

The algorithm for detecting cookie paywalls makes use of the NLP library NLTK [28]. The detection occurs in two steps: text preprocessing and feature extraction.

Text preprocessing begins by searching for and replacing notions of a fee with a fixed token. The search finds price tags in a text using a regular expression searching for a number that is led by or followed by a currency. For example: "1.9 SEK", "100 kr", "EUR 10.0", "13 €", "£20.1". If a price tag is found, it is replaced with the token "¤cpprice¤ ", which can be treated as any other token during feature extraction. The regular expression is built using the built-in *Regular Expression Operations* (re) library for Python, which we used in a function as shown below:

```
import re
cp_price = "¤cpprice¤ "
curr = "(\€|eur|\$|usd|sek|\\\kr|cad|chf|dkk|gbp|\£|hrk|\\\kn|isk|nok|nzd|try)"
price = "\d+\.?\d*"

def find_and_replace_price(text):
    text = re.sub(f'{curr}\W*{price}', cp_price, text.lower())
    text = re.sub(f'{price}\W*{curr}', cp_price, text.lower())
    return text
```

Afterward, the text is tokenized, which separates words by special characters such as whitespace, punctuation marks, and commas. The tokenization is performed using the `word_tokenize()` from NLTK.

After tokenization, stop words are removed. Stop word removal allows phrases from Listing 1 such as Item 2 and Item 5, to be treated equally by removing tokens such as "any". The list of stop words is obtained from a built-in corpus in NLTK, which is used to remove them from the text.

The final step of text preprocessing is stemming, which for this project was done using NLTK's PorterStemmer. The use of stemming allows different inflections of words to be treated equally which simplifies feature extraction further. Examples of how some words can be stemmed can be seen below. This step also converts all words to lowercase, allowing different capitalizations of words to also be treated equally.

1. "Subscriber", "subscribe", "subscribing": "subscrib"
2. "Cookies", "cookie": "cooki"
3. "Advertising", "advertiser", "advertisement": "advertis"
4. "Tracking", "tracker": "track"

Since the tokens, after stemming and stop word removal, retain their order as well as separation by periods and commas, we can perform effective word and n-gram

1. “no advertising cookies” (ex. `lachainemeteo.com`)
2. “without advertising tracking” (ex. `vienna.at`)
3. “without ad tracking” (ex. `zeit.de`)
4. “without tracking, external banner and video advertising” (ex. `heise.de`)
5. “without any advertising tracking” (ex. `express.de`)
6. “without advertising banners” (ex. `gutekueche.at`)

Listing 1: Phrases typical of cookie paywalls

analysis (described in Section 2.5). First and foremost, sites that employ cookie paywalls likely have the word “cookie” in them, and the majority have some inflection of the word “track”. As for phrases, there are a few that are particularly unique to cookie paywalls. “Refuse and subscribe” is a phrase only found on French websites so far (originally found on `lemonde.fr`). All sites found with this phrase employed a cookie paywall. As for less reliable, but more widely used phrases, there are a number of variations of the phrase “without advertising cookies”. Some, but not all, variations can be found in Listing 1.

This list is not exhaustive, but it illustrates that it may not be effective to try to come up with different static variations of phrases that have the same meaning, which is why stemming and stop word removal is useful.

5.1.4 Reporting Crawler Results

Once a page has been scraped, parsed, translated, and recognized, the result is stored in a database. A program has been developed that can use both Postgres and SQLite3. However, SQLite databases cannot be used when multiple crawlers are working on the same database, so Postgres is used in the scalable implementation (see Section 5.2). The results are stored in a single table, whose schema is shown in Table 3 in Appendix A.

5.2 Scaling the Crawler

In order to achieve the cluster configuration as described in Section 4.4, a number of technologies has been used, including containerization using Podman, container orchestration software Kubernetes, ACID-compliant database Postgres, and automated browser tests with Selenium.

5.2.1 Containerizing the Crawler

To consistently replicate crawlers in a cluster, we have chosen to put the Crawler inside a container image using Podman. This allows for the deployment of the Crawler on any machine with Docker or Podman installed. More importantly, it allows the Crawler to be replicated in a computing cluster, as explained in the next section.

A Docker image can be built using instructions from a ‘Dockerfile’. Instructions

can include what image to base this image on, commands to run, files to copy, environment variables to set, and more. The Dockerfile used for this project is shown in Listing 3 in Appendix B.

5.2.2 Configuring the Kubernetes Cluster

The container orchestration software Kubernetes was used to manage crawlers, storage, the delegator, and the database. To develop and test our Kubernetes configuration, we used a Virtual Private Server (VPS) on Amazon Lightsail, on which we installed Microk8s. Microk8s is a way of setting up a Kubernetes cluster, along with key components like DNS, a local registry, and more, with just a few commands. Our particular VPS was running Debian Bullseye, but any operating system with Snapd installed can run a Microk8s cluster.

5.2.3 Database Setup

In order for the crawlers to be able to fetch URLs and store results, data storage is needed. We chose the database PostgreSQL, due to its maturity and large community [51], and due to its ACID compliance [52].

In order to set up credentials concerning the database, we made a ConfigMap as seen in Listing 4 in Appendix B, which allows Kubernetes deployments to make use of these values when setting up containers.

The values from the ConfigMap will be applied as environment variables for the Postgres and crawler deployments. But first, the database needs persistent storage. By default, all containers use ephemeral storage, meaning when they are stopped, all data is lost. Therefore, we set up persistent storage by making a PersistentVolume and a PersistentVolumeClaim, as seen in Listing 5 in Appendix B.

The PersistentVolume creates a volume with 75Gi of data on */mnt/data* on the host machine. The PersistentVolumeClaim allocates all of this storage and sets the read-write mode.

The ConfigMap values and volume are then used in the Postgres deployment, shown in Listing 6.

In order for this database to be reachable through the network to the crawlers', a Service is needed, as specified in the YAML file in Listing 7. This configuration file, once applied with `kubectl apply -f <filename>.yaml`, will open port 5432 for all pods within the same namespace.

5.2.4 Selenium Hub Setup

In order for the crawlers to have access to Selenium WebDrivers, there are generally two alternatives concerning containerized Selenium instances. There exists a comprehensive effort in providing Selenium Grid in Docker container format [53], which we have made use of. They provide a standalone WebDriver container image

for each browser, and the alternative of running a Selenium Grid Hub, as detailed in Section 2.3.1.

The YAML file in Listing 8 in Appendix B shows our Selenium Hub Kubernetes deployment. The YAML file shown in Listing 9 opens port 4444 so that other pods in the same namespace can access it. Once these configuration files are applied using Kubectl, browsers can be connected. The example in Listing 10 in Appendix B deploys 16 Firefox browsers to the Hub.

The computing resources assigned to each deployment is crucial because if the browsers don't have enough memory or processing power, they become prone to crashing or bottlenecking.

Once these configurations are applied, the crawlers can connect to the Hub by requesting WebDriver sessions to the hostname 'selenium-hub' and port 4444.

5.3 Website rankings list

The sample of 1 million URLs to crawl has been selected using a website ranking made by Tranco. Tranco is a research-oriented top-sites ranking effort by Le Pochat *et al.* [54], which contains a freely downloadable list of the top 1 million most popular websites. In their study, they argue that other popular site rankings, like Alexa, are easily manipulated and even untrustworthy due to commercial, for-profit incentives. Another argument as to why such rankings may be unreliable is that their methods are not always disclosed. This hinders research to be made on behalf of the validity of the rankings. The authors also state that many large popular site rankings (Alexa, Cisco Umbrella, Majestic, and Quantcast) hardly agree on any single domain ranking, and even allow malicious sites to be included. The authors additionally proved that single HTTP requests can affect thousands of rankings, and pushing a domain into the top 10,000 is straight-forward. Additionally, rankings like Alexa average sites only during a day's time frame, meaning the rankings can change drastically from day to day.

The authors of the Tranco paper state that their list is more resistant to manipulation and open as to how it is produced. These reasons are why we have chosen it for this project. Their list is downloadable from `https://tranco-list.eu`. For our particular tests, we used the Daily List of the top 1 million sites, downloaded on May 18th of 2023, from `https://tranco-list.eu/list/PZPLJ/1000000`.

Chapter 6

Tests and Results

This chapter describes the various procedures and results regarding cookie paywalls and the performance of the crawler. We start with the hardware and software specifications that the crawler ran on in Section 6.1. The results of this research consist mainly of two parts, cookie paywalls and crawler performance. Regarding cookie paywalls, we reported and discussed how many we found and their associated data in Section 6.2. In terms of the accuracy of our detector, we have considered the true positives out of all the positives from the crawler's results. In terms of performance, we consider throughput, i.e. the number of pages the crawler is able to process in a given period of time. In terms of scalability, we consider how throughput, CPU usage, and memory usage increase with the size of the cluster, which is reported in Section 6.3.

6.1 Hardware and Software Configuration

The same configuration was used for both the large-scale crawl and the benchmark tests. We used the AWS Elastic Kubernetes Service (EKS) to deploy a Kubernetes cluster under which we created a node group. Node groups provision nodes for the Kubernetes cluster, and it is possible to choose between a variety of hardware capabilities. Our chosen node group consisted of a single node of the "m6i.32xlarge" instance type, which had 128 virtual CPUs, 512 GB of memory, 50 Gbps of network bandwidth, and 1 TB of storage. We chose a single node because there was no need for redundancy in terms of high availability, and we would only be running the crawler for a few days. We did confirm that the Crawler functions as expected when running on two nodes in a node group, where each node had half the computing resources as the one previously described.

The node ran Amazon Linux 2 (AL2_x86_64), which according to Amazon is optimized for EKS instances [55]. The AMI release version used was '1.26.2-20230509' and the Kubernetes Server version used was 'v1.26.4-eks-0a21954'. Kubernetes client (Kubectl) versions used varied, but one of the working versions was 'v1.27.2'.

As for Docker images used, Postgres was pulled from Docker Hub using the official

Postgres image [56] with the tag 'postgres:15.2'. All Selenium components had tag version '4.9.1-20230508'. That is, 'selenium/hub:4.9.1-20230508' for the Hub and 'selenium/<browser name>4.9.1-20230508' for the browsers, where '<browser name>' is replaced by 'chrome', 'firefox', 'safari' or 'edge'.

Manual verification of cookie paywalls was performed using the following browser and operating system versions:

- Firefox Version 113.0.2 (64-bit) incognito on MacOS
- Firefox Version 113.0.2 (64-bit) incognito on Linux (Flatpak)

6.2 Cookie Paywall Results

This section presents the results concerning cookie paywalls. In the early stages of this project, we ran a beta version of the crawler program on a small portion of the Tranco list, as detailed in Section 6.2.2. In the final stage, we ran a large-scale crawl on all 1 million sites from the Tranco list.

6.2.1 Tests and Verification of Cookie Paywalls

To perform the intended large-scale crawl, the scalable crawler was configured to run 32 crawlers on the top 1 million sites, using a matching 32 Firefox browsers. The performance of this crawl was determined by page throughput, how many URLs resulted in errors, and what type of errors there were.

Once the crawl was complete, the cookie paywall sites flagged as 'likely' were manually confirmed, along with a classification of geographical basis, website category, and paywall price per month. These details were recorded in a spreadsheet.

The country in which the site is based was determined by parsing the WHOIS requests for each site. Responses typically contain one or more fields whose names include the word "country". The top field whose name matched this criterion was then extracted for each site and resolved to a country.

Website type was determined by Cyren's URL Category Checker [57]. Cyren is a cybersecurity company that offers a number of products related to malware, cyberattacks, Internet spam, and more. Their products make use of a URL classifier to assess threats from web pages, which is provided on their website: `https://www.cyren.com/security-center/url-category-check-gate`. They claim to provide an API for programmatically checking website categories, but we couldn't register for this API because we got an error when filling out their form. However, since it was still possible to manually paste URLs into their web page, we recorded the resulting category for each URL and added it to the spreadsheet. The categorizer sometimes returned two categories for the same URL. Only the first category was included in this report.

The method for confirming cookie paywalls was similar to that used by Morel *et al.* [9], where three annotators individually classified the results before discussing any

disagreements verbally. Our confirmation was performed by two annotators who each browsed their assigned half of the likely cookie paywall URLs, with results noted in the spreadsheet.

6.2.2 Heuristic Crawling Results

The initial stages of this thesis were spent developing a crawler without initial regard for running in a cluster of crawlers. The goal was to find a few more cookie paywalls with phrasings differing from the ones found by Morel *et al.* [9]. The early-version crawler found a total of 77 cookie paywalls by crawling roughly 40,000 websites and manually confirming the ones reported as 'likely' by the program. By browsing through these, we identified a CMP, called 'Contentpass', which offers a cookie paywall solution to all its partners. They have posted all their alleged partners at `https://www.contentpass.net/publications`. Through this CMP, we were able to identify an additional 191 sites, for a total of 271 cookie paywalls.

6.2.3 Large-scale Crawling Results

The Crawler processed 1 million pages in about 5 days, reporting 330 likely cookie paywalls. The confirmed total was 323, giving the Crawler a true positive accuracy of ~98%. The average throughput of this crawl was ~2.3 *pages/second*.

Of the 1 million pages, 322,542 resulted in an error, with the error distribution shown in Figure 6.1. Timeout errors make up the largest portion of errors, which happen when a page doesn't load in under 8 seconds.

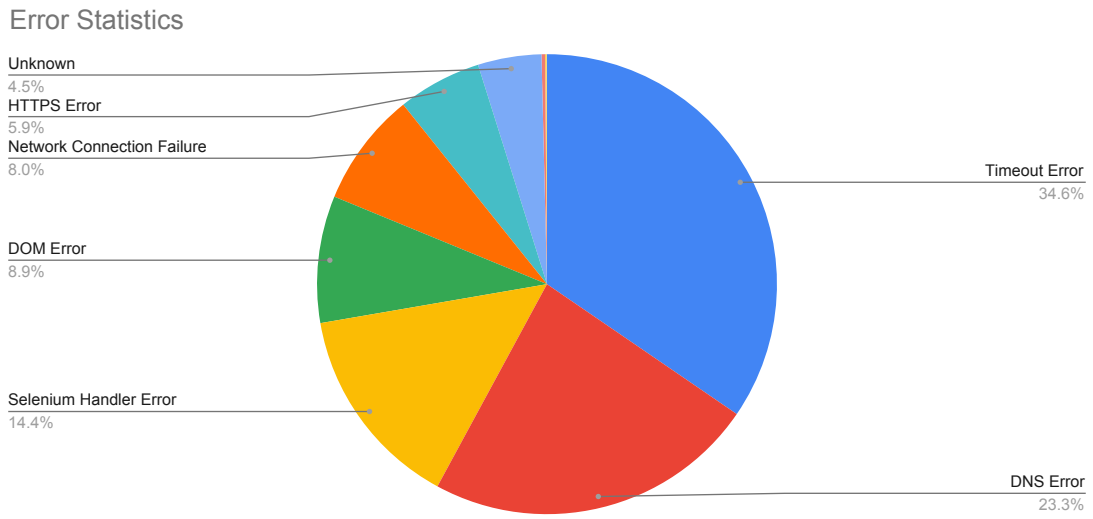


Figure 6.1: The distribution of errors when running the Big Crawl. The total count of errors is 322,542.

108 cookie paywalls from the previous heuristic methods as detailed in Section 6.2.2, were not found in the large-scale crawl, but were combined with the 323, making a total of 431 (listed in Table 4 in Appendix C). Of the cookie paywalls found during

the large-scale crawl, the majority were found in Germany, with France being the second most common country. A graph of the distribution of cookie paywalls across all countries is shown in Figure 6.2. Only 8 out of 431 cookie paywalls were based outside of Europe.

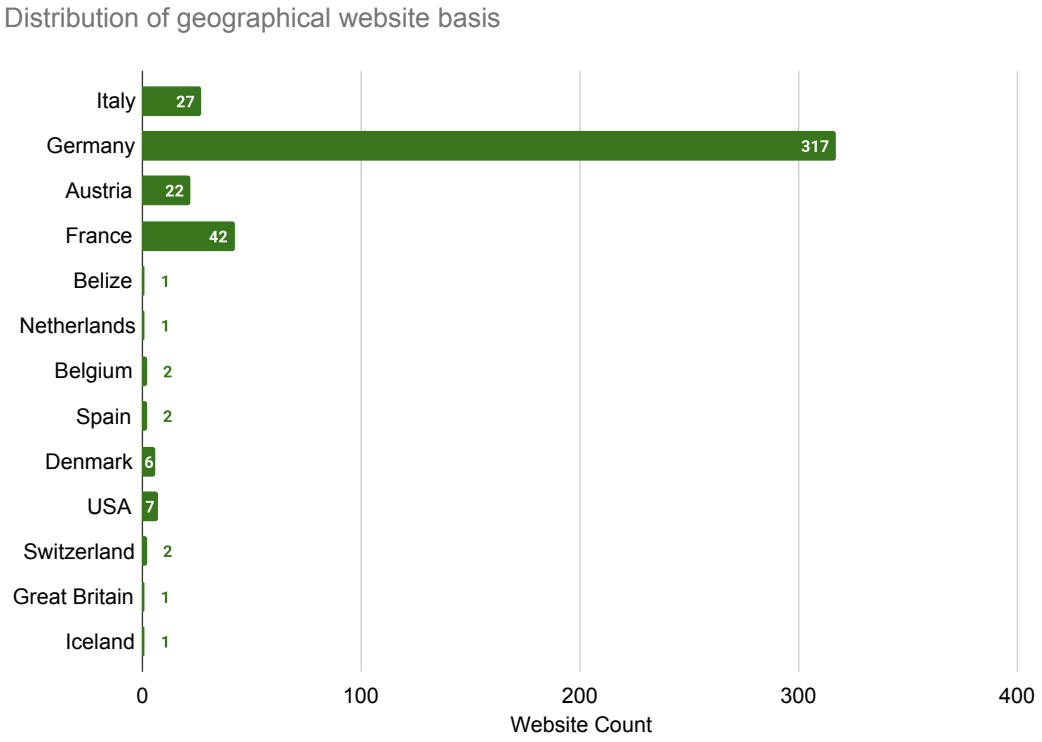


Figure 6.2: Results per country from large-scale crawl

The distribution of website categories shows that a large number of the found cookie paywalls were classified as News. The three categories News, Business, and Computers & Technology together represented a majority of all cookie paywalls. The full distribution of the results is shown in Figure 6.3.

For each cookie paywall, a monthly price was noted. The distribution of price is visualized in Figure 6.4 with an average price of €3.34 per month. Only three sites had a price above €10, with one notable site with a monthly price of €49 per month left out of the chart for readability. Temporary starting and commitment discounts were not considered.

The different classifications of site types that were reported by Cyren can be found in Table 1 in Appendix A.

6.3 Performance & Scalability

This section details the results of how the Crawler performed in terms of performance and scalability. For this project, performance refers to how many web pages are processed by the Crawler during a period of time. Scalability refers to how performance,

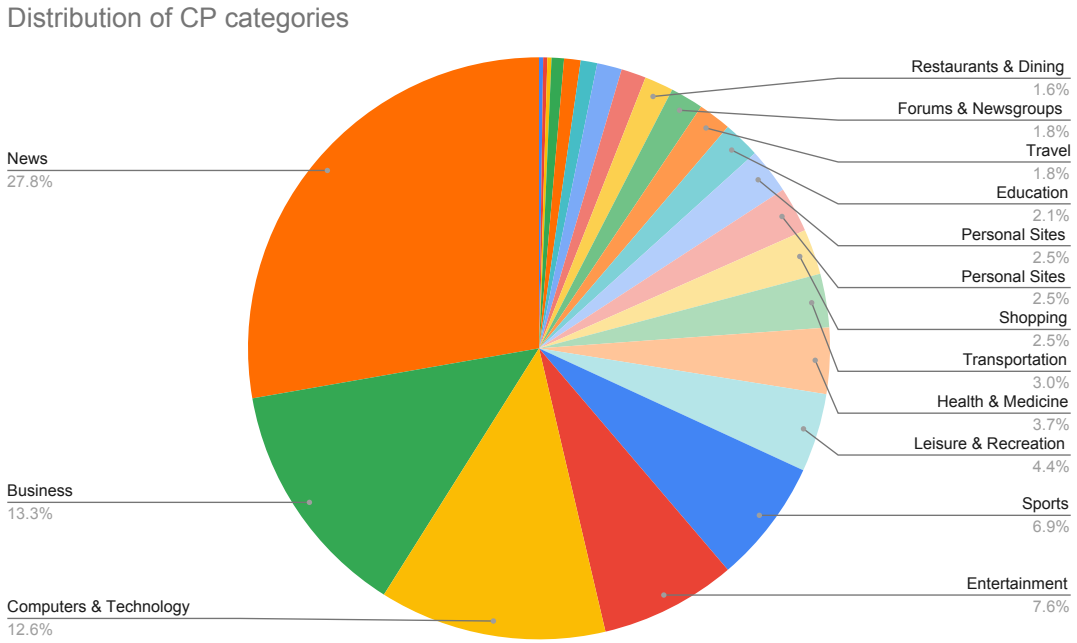


Figure 6.3: Distribution of website categories as reported by Cyren [57]

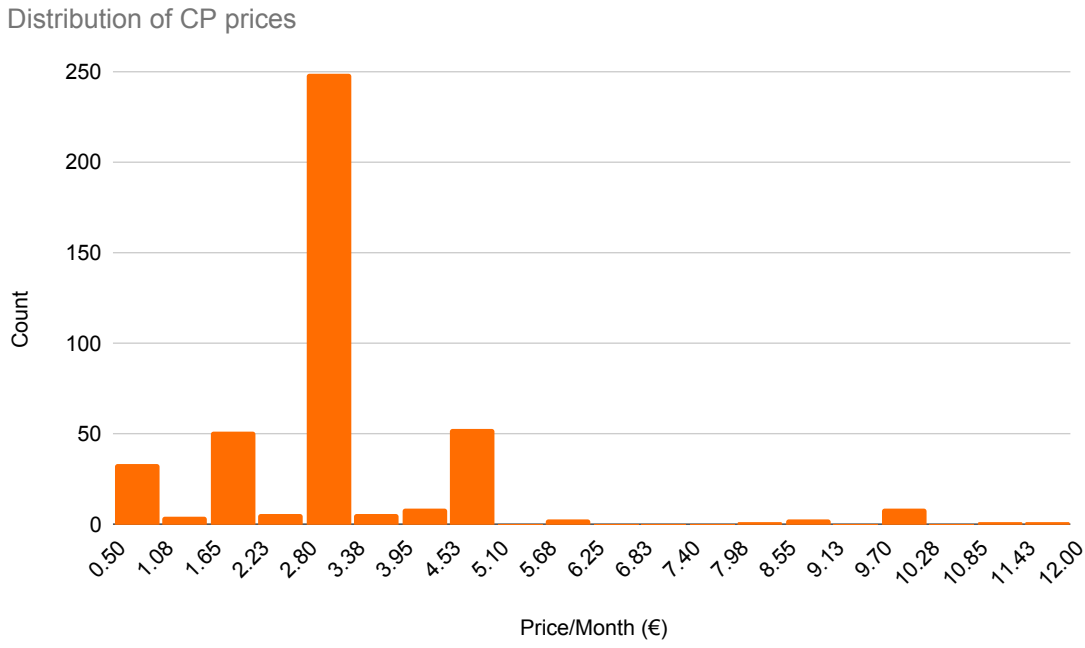


Figure 6.4: Distribution of cookie paywall prices in euros per month

memory, and CPU usage change along with the number of crawlers.

Memory usage and CPU usage are measured as they are reported by the Kubernetes Metrics Server, by running `kubectl top node`, which reports the CPU and memory

allocated on the node(s) on which Kubernetes is running.

6.3.1 Scalability Tests

For the Crawler to be 'scalable', performance must increase at least linearly with the number of crawlers, while CPU and memory usage must increase at most linearly. To measure scalability, we measured how performance develops when running 1, 2, 4, 8, 16, and 32 crawlers in one cluster, while also noting CPU and memory usage. The 1000 URLs to be crawled were randomly sampled using Python's built-in `random.sample(population, k)` on the list of 1 million websites found in the Tranco list we downloaded (see Section 5.3). The same random sample was used for all the benchmarks, which consisted of launching the Crawler and Selenium deployments in Kubernetes and letting them process the sample of websites. In addition to throughput, we have added details about crashes, errors, and results.

The testing process is shown in detail in Algorithm 5.

Algorithm 5 Testing processes for performance and scalability

$N \leftarrow 4$
$S \leftarrow Random\ sampling\ of\ 1000\ URLs\ from\ Tranco$
$Results \leftarrow dict()$
while $N \leq 32$ **do**
    Deploy Selenium Hub with N Firefox browsers
    Flag S for crawling
    Deploy N crawlers
    while $\nexists U_{crawled} \in S$ **do**
    end while
    Begin timer
    while $\neg(\forall U_{crawled} \in S)$ **do**
        Log: $time_{elapsed}, pages/second, CPU, MEM, len(U_{uncrawled} \in S), ...$
        Wait 1 second
    end while
    $N \leftarrow N * 2$
    Delete crawler and Firefox deployments
end while
Present logged results in the report

To summarize the benchmarking process, an iteration of the benchmark delegates 1000 pages for crawling and deploys N browsers to the Selenium Hub along with N crawlers. The test then waits until the first result is reported by any crawler and then proceeds to measure. First, information is gathered on throughput, remaining URLs, CPU usage, memory usage, and running crawlers. Then, a 10-second wait is initiated. In the real script, this 10-second wait is offset by how long it took to perform measurements so that each iteration effectively takes 10 seconds. The measuring process is repeated until all URLs are crawled, after which the deployments are deleted.

6.3.2 Throughput

An important metric for determining the scalability of the Crawler is the throughput as a function of the number of crawlers. The output of this function is illustrated in Figure 6.5a. The graph shows that the average throughput increases linearly with the number of crawlers. From 1 to 32 crawlers, the final average throughput increases from 0.2 to 7.46 pages per second.

Figure 6.5b shows the throughput for each benchmark, with a logarithmic horizontal axis. As the average throughputs settle at the end, the distance between each line is roughly equal. This further illustrates that as the number of crawlers is doubled, so is throughput. A more detailed sheet of the results of the benchmarks can be seen in Table 6.1.

Table 6.1: Performance and scalability results over the number of crawlers with throughput denoted as p

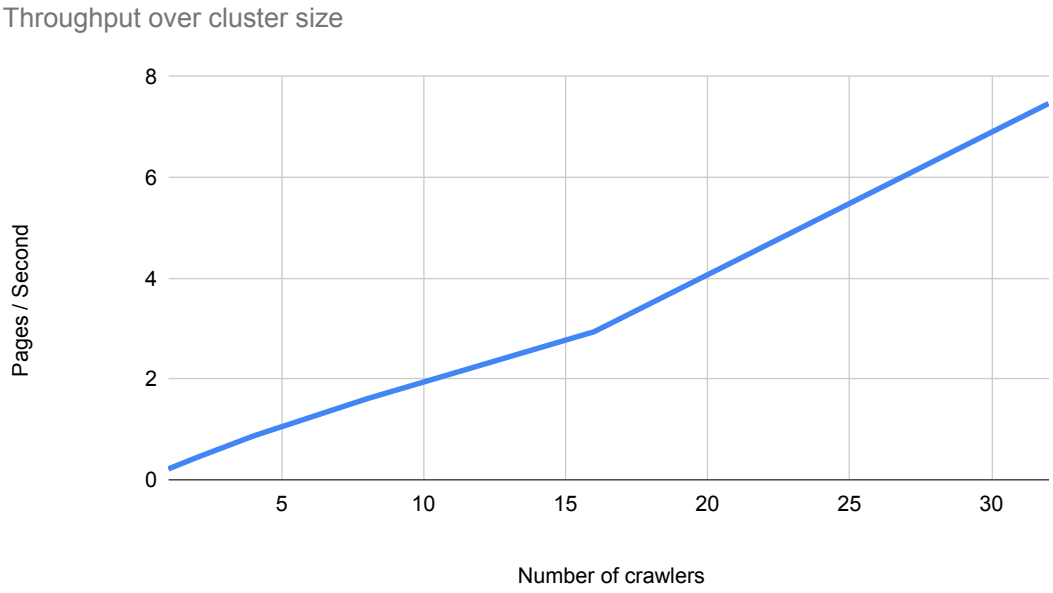
Crawlers	**Avg.** p	**CPU%**	**Memory%**	**CPU%**/p	**Mem%**/p
1	0.216	1%	0%	0.05	0.00
2	0.442	2%	1%	0.05	0.02
4	0.870	4%	1%	0.05	0.01
8	0.161	8%	2%	0.05	0.01
16	2.987	17%	3%	0.06	0.01
32	7.460	31%	5%	0.04	0.01

6.3.3 Memory and CPU Usage

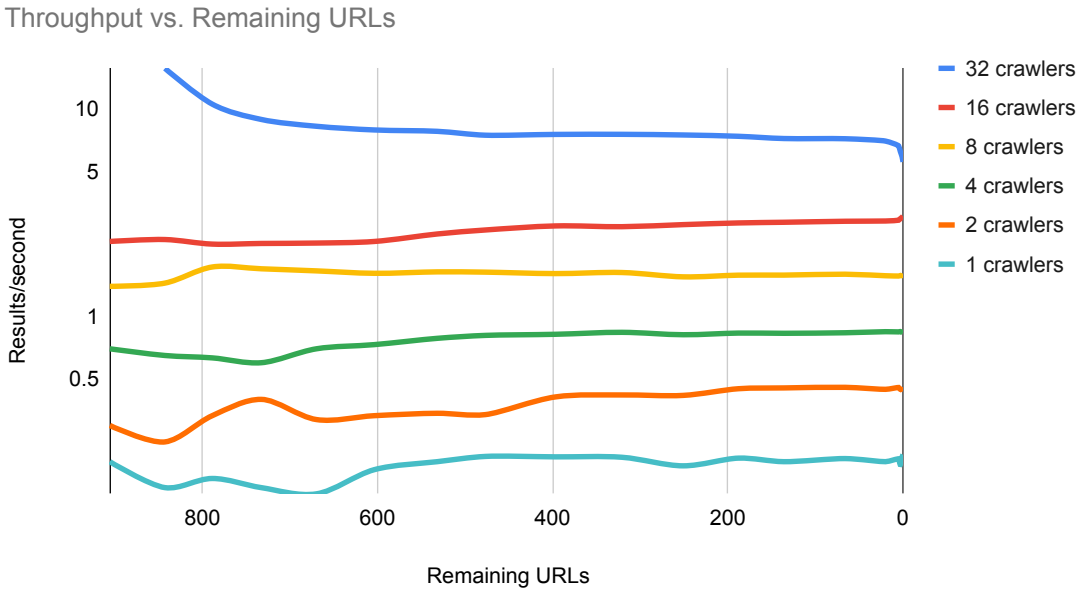
Percentage CPU usage and percentage memory usage have been measured in relation to performance, the results of which are shown in Figure 6.6.

CPU usage for the benchmarks is illustrated by two lines, where the solid one shows the percentage of CPUs used and the dashed one shows CPU usage per throughput. CPU usage increases linearly throughout, indicating that there are no CPU-related bottlenecks. CPU percentage over throughput is roughly constant, which shows that the Crawler can be considered scalable in terms of CPU and memory usage.

Memory usage is similarly represented by two lines. One line shows the percentage of available memory that is being used, while the other shows memory usage over throughput. Looking at the trends in memory usage, a linear increase can be seen in the percentage of memory used over the number of crawlers, indicating scalability in terms of memory. This is further illustrated by the dashed line for memory over throughput, which remains roughly constant, meaning more performance does not exponentially increase demands for memory.



(a) Average throughput over the number of crawlers



(b) Average throughput as 1000 URLs are crawled for different numbers of crawlers

Figure 6.5: Indications of scalability

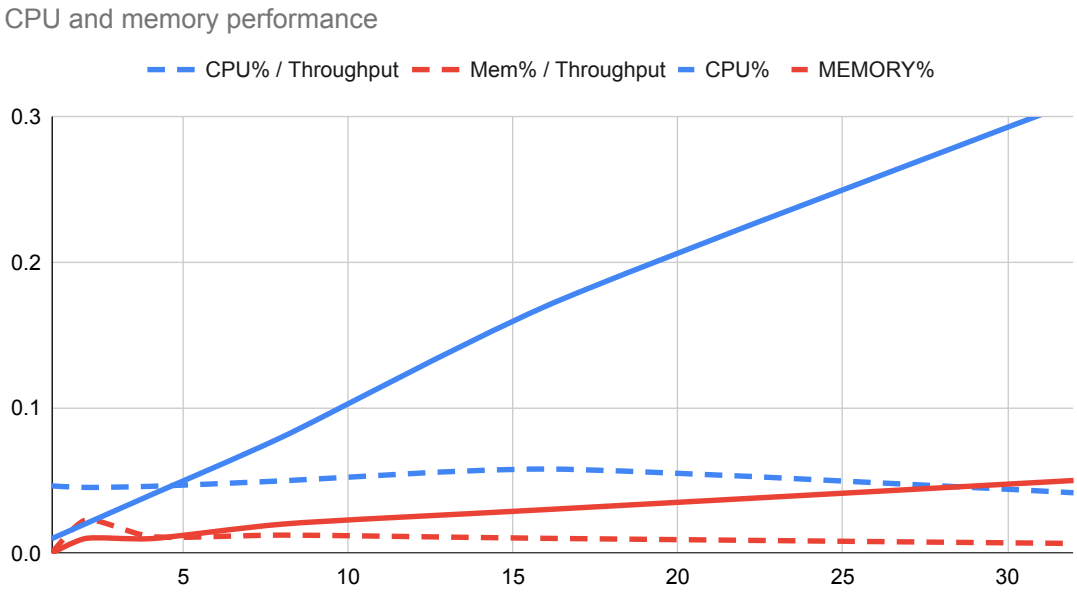


Figure 6.6: Average CPU and memory usage in percentage and compared to throughput

Chapter 7

Discussion

This chapter discusses results and implications of our research on, large-scale cookie paywall detection.

7.1 Cookie Paywalls

This chapter includes analyses and discussions about cookie paywalls and the roles they may play in online publishing. We identified 431 instances of cookie paywalls on various websites, along with their geographical basis, category, and price. Based on these findings, this chapter explores the impact of cookie paywalls on user privacy and discussions on their geography, categories, and price.

7.1.1 Geography and Lawfulness

Our results show an outnumbering prevalence of cookie paywalls on in Germany. This finding is particularly relevant to RQ2, given the position taken by the German Data Protection Authority (DPA) which claims that cookie paywalls violate freely given consent [9]. This indicates that the prohibition of cookie paywalls may not currently have an affect on their prevalence.

One possible contributor to the concentration of cookie paywalls in Germany may be the Consent Management Platform (CMP) known as Contentpass. Unlike other CMPs that offer both banner and paywalls, this German-based CMP only offers a cookie paywall solution. However, we found 317 cookie paywalls based in Germany, while Contentpass claims to have 220 partners, meaning 97 are not using Contentpass. 97 is still considerably more than the runners-up France, Italy, and Austria, meaning Contentpass may not be the only reason for the prevalence in Germany.

The majority of cookie paywalls detected in our study are based in Europe, with only 8 being found outside, further illustrated by the heatmap in Figure 1. This may be due to a biased methodology. Our detection algorithm was based on a previous study that specifically focused on identifying European cookie paywalls. Therefore, it is possible that our algorithm's preference for European sites had a strong influence on the results. This bias in our data sample raises questions about the distribution

and prevalence of cookie paywalls worldwide, especially outside of Europe. To gain a deeper understanding, future research should examine different geographic regions and develop detection algorithms that are less specific to certain regions.

7.1.2 Categories

27.4% of confirmed cookie paywalls identified as “News” by Cyren. This may indicate a particular monetary reliance of targeted advertising for newspapers.

In addition, the frequency of cookie paywalls on sites in the Business (13.2%), Computer & Technology (12.3%), and Entertainment (7.7%) categories further suggests a potential reliance on selling user data for ad space in these sectors as well. These categories include sites that often host high-traffic platforms that attract a large user base that can be leveraged for targeted advertising purposes.

By implementing cookie paywalls, these sites may be incentivizing users to consent to be tracked, allowing them to gain valuable insights for targeted advertising campaigns. It is important to note, however, that further research is needed to establish a definitive correlation the presence of cookie paywalls and the reliance on selling user data for ad space within these specific website categories.

7.1.3 Price

By analyzing the prices found for each cookie paywall, we found that the vast majority (67%) cost between €2 and €4 per month. Morel *et al.* [9] also made an analysis of the costs related to the legality of cookie paywalls. They found that one criterion in the case-by-case assessment by DPAs is that the cost of the paywall should be "reasonable" or "fair remuneration". However, there is no mention of what a reasonable price might be.

The prices from our results range from €0.75 to €49 per month, with an average of €3.34. Almost all sites were offering a monthly subscription and only a few offered weekly or annual subscriptions. Among the cookie paywalls found, all used a subscription-based payment model.

It is worth noting that Contentpass offers a subscription-based model of €2.99 per month that gives access to several of their partners. This means that paying this subscription on one site gives access to several other sites. This could be compared to services like the audiobook platform Storytel and the music streaming platform Spotify, which give access to a library of content from different creators under a single subscription.

7.2 Developing a Scalable Crawler

We were able to build a scalable crawler that was performant enough to crawl a million sites in less than 5 days. However, we later found that the Selenium Hub acted as a performance bottleneck. It was bottlenecking because we initially provided the Hub with insufficient computing resource limits. We increased CPU and memory

limits for the Hub which lead to a tripling of throughput when deploying higher number of crawlers. Running the crawl without the bottleneck in the otherwise same configuration would have allowed it to finish in under 2 days.

The results of the benchmarks, as can be seen in Table 6.1, show that the average CPU utilization never exceeded ~31% and the average memory utilization never exceeded ~5%. This means there were ample resources in terms of both CPU and memory. Doubling the cluster size, meaning performance would be doubled as well, would have allowed the 1 million crawl to finish in under a day.

7.3 Future Work

This master's thesis has succeeded in providing insight into the prevalence of cookie paywalls. However, as with any research, there are potential areas of improvement for future work that could improve understanding of this topic. In this chapter, we suggest some possible directions for future research, building on our findings and acknowledging the limitations highlighted in the previous chapters. These suggestions are intended to guide further investigation, address unanswered questions, and provide valuable insights into related areas.

Two obvious approaches to finding more cookie paywalls are to crawl more sites and to improve the detection algorithm. To crawl more sites, it is possible to simply run our crawler for a longer period of time on a larger number of websites. Alternatively, the scalability issues discussed in Section 7.2 can be investigated and solved, allowing the Crawler to better utilize available computing resources. To improve sustainability in the Crawler, it could be improved to be more efficient, inherently increasing throughput. Because the crawler program performs a number of different and generally independent tasks, many of them could theoretically be pipelined. For example, while a crawler is loading one URL, another URL could be translated and another URL could be extracted for features.

There is potential to improve the flexibility of the detection algorithm by exploring the use of machine learning. By leveraging additional insights from this research, a training dataset can be created that includes a wide range of sites with different cookie paywall implementations. Machine learning algorithms, such as supervised learning or deep learning models, could potentially be applied to improve detection accuracy and reduce bias. Incorporating machine learning has the potential to improve precision and recall rates, as well as properly expand the scope of detection to additional geographic regions.

Another avenue for future work is to build on the results of our research and conduct further studies to analyze trends in the prevalence and characteristics of cookie paywalls. While our research provides current insights, it only represents one snapshot in time. By periodically repeating the large-scale analysis, it would be possible to track the trend of cookie paywalls over time. This trend analysis could shed light on the effectiveness of regulatory measures, industry practices, and user awareness campaigns in shaping the adoption and persistence of cookie paywalls. In

addition, examining the influence of technological advances would provide a better understanding of the dynamics between paywall strategies and internet privacy.

Chapter 8

Conclusion

Finding cookie paywalls is akin to looking for a needle in a haystack. By crawling 1 million web pages, only 323 were found and confirmed, meaning that they represent at least 0.0323% of the top 1 million most popular web pages. 108 cookie paywalls were identified heuristically, bringing the total number of confirmed cookie paywalls to 431, which are listed in Table 4 in Appendix C. Geographically, cookie paywalls seem to be disproportionately prevalent in Germany, with a count of 317. A possible contributing factor is the prevalence of Contentpass, a consent management provider, which claims to have 220 partners using its product. However, that leaves 97, which is still considerably more than other regions. The prevalence in Germany is particularly interesting since cookie paywalls are prohibited by law there, as discussed in Section 7.1.1. The next most common regions for cookie paywalls are France, Italy, and Austria.

The most likely employers of cookie paywalls are seemingly various types of newspapers and magazines. All known cookie paywalls rely on subscription-based payment models, which seems to fit well with newspapers. Paywall costs across all 431 paywalls average €3.34 per month. However, prices ranged from €0.75 in some cases, and up to €49 in one particular case.

Developing the Crawler responsible for gathering most of this data was built using primarily Python along with the Selenium Web Testing Framework. The Crawler ran in a cluster by packaging the crawler program into a Docker container image and replicating with it Kubernetes. The cluster ran 32 crawlers on an EKS cluster and processed 1 million pages in roughly 5 days with throughput averaging roughly 2.3 *pages/second*. After noticing insufficient CPU and memory limits set on the Selenium Hub deployment, we provided more resources for this component. This resolved a bottleneck and lead to a throughput of 7.46 *pages/second* when running the benchmarks.

Bibliography

[1] R. Koch. "Cookies, the gdpr, and the eprivacy directive." en-US. (May 2019), [Online]. Available: https://gdpr.eu/cookies/.

[2] M. Degeling, C. Utz, C. Lentzsch, H. Hosseini, F. Schaub, and T. Holz, "We value your privacy ... now take some cookies: Measuring the gdpr's impact on web privacy," *CoRR*, vol. abs/1808.05096, 2018. arXiv: 1808.05096. [Online]. Available: http://arxiv.org/abs/1808.05096.

[3] C. Matte, N. Bielova, and C. Santos, "Do cookie banners respect my choice? : Measuring legal compliance of banners from iab europe's transparency and consent framework," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 791–809. DOI: 10.1109/SP40000.2020.00076.

[4] Ø. H. Kaldestad, "Companies use design to take our time, money and personal data," Dec. 2022. [Online]. Available: https://www.forbrukerradet.no/side/companies-use-design-to-take-our-time-money-and-personal-data/ (visited on 12/02/2022).

[5] B. Wessel-Tolvig, "What are dark patterns in cookie banners?," Aug. 2022. [Online]. Available: https://cookieinformation.com/resources/blog/what-are-dark-patterns-in-cookie-banners/ (visited on 12/02/2022).

[6] "Report of the work undertaken by the cookie banner taskforce." (), [Online]. Available: https://edpb.europa.eu/our-work-tools/our-documents/other/report-work-undertaken-cookie-banner-taskforce_pt (visited on 02/08/2023).

[7] Lee Rainie, Sara Kiesler, Ruogu Kang, and Mary Madden. "Anonymity, privacy, and security online." (Sep. 5, 2013), [Online]. Available: https://www.pewresearch.org/internet/2013/09/05/anonymity-privacy-and-security-online/ (visited on 04/11/2023).

[8] Mary Madden. "Public perceptions of privacy and security in the post-snowden era," Pew Research Center: Internet, Science & Tech. (Nov. 12, 2014), [Online]. Available: https://www.pewresearch.org/internet/2014/11/12/public-privacy-perceptions/ (visited on 04/11/2023).

[9] V. Morel, C. Santos, Y. Lintao, and S. Human, "Your consent is worth 75 euros a year - measurement and lawfulness of cookie paywalls," ACM, Nov. 2022. DOI: 10.1145/3559613.3563205. [Online]. Available: https://doi.org/10.1145%5C%2F3559613.3563205.

[10] C. C. Lukasz Olejnik Tran Minh-Dung, "Selling off privacy at auction," 2013, hal-00915249.

[11] F. Coton and V. Ruelle, "The end of third-party cookies: Nothing but smoke and mirrors if the RTB winner takes it all?" In *Time to reshape the digital society*, ser. Collection du CRIDS, Bruxelles: Larcier, 2021, pp. 209–233.

[12] "General data protection regulation (GDPR) – official legal text," General Data Protection Regulation (GDPR). (), [Online]. Available: `https://gdpr-info.eu/` (visited on 01/27/2023).

[13] *HTTP | MDN*, en-US, Mar. 2023. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP` (visited on 04/27/2023).

[14] "HTML: HyperText markup language | MDN." (Mar. 13, 2023), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTML` (visited on 04/27/2023).

[15] J. Krause, "Shadow DOM," in *Developing Web Components with TypeScript: Native Web Development Using Thin Libraries*, J. Krause, Ed., Berkeley, CA: Apress, 2021, pp. 43–52, ISBN: 978-1-4842-6840-7. DOI: `10.1007/978-1-4842-6840-7_3`. [Online]. Available: `https://doi.org/10.1007/978-1-4842-6840-7_3` (visited on 04/28/2023).

[16] "JavaScript | MDN." (Apr. 5, 2023), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTML` (visited on 04/27/2023).

[17] "Asynchronous JavaScript - learn web development | MDN." (Feb. 23, 2023), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous` (visited on 04/27/2023).

[18] "Using HTTP cookies - HTTP | MDN." (Apr. 10, 2023), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies` (visited on 04/27/2023).

[19] *Set-Cookie - HTTP | MDN*, en-US, Apr. 2023. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie` (visited on 05/02/2023).

[20] "Web browser," Web Browser. (Apr. 11, 2018), [Online]. Available: `https://web.archive.org/web/20211206205907/https://webbrowsersintroduction.com/` (visited on 04/11/2023).

[21] *Web scraping*, in *Wikipedia*, Page Version ID: 1147883001, Apr. 2, 2023. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Web_scraping&oldid=1147883001` (visited on 04/11/2023).

[22] "About selenium." (), [Online]. Available: `https://www.selenium.dev/about/` (visited on 02/21/2023).

[23] "Browser market share worldwide," StatCounter Global Stats. (), [Online]. Available: `https://gs.statcounter.com/browser-market-share` (visited on 04/27/2023).

[24] *Components of Grid 3*, en, Section: documentation. [Online]. Available: `https://www.selenium.dev/documentation/legacy/selenium_3/grid_components/` (visited on 05/23/2023).

[25] "What is a container? | microsoft azure." (), [Online]. Available: `https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-container` (visited on 04/12/2023).

[26] *Open Container Initiative - Open Container Initiative*. [Online]. Available: `https://opencontainers.org/` (visited on 05/23/2023).

[27] Sarah Conway. "Kubernetes is first CNCF project to graduate," Cloud Native Computing Foundation. (Mar. 6, 2018), [Online]. Available: https://www.cncf.io/blog/2018/03/06/kubernetes-first-cncf-project-graduate/ (visited on 04/12/2023).

[28] "NLTK :: Natural language toolkit." (), [Online]. Available: https://www.nltk.org/index.html (visited on 02/16/2023).

[29] P. Willett, "The Porter stemming algorithm: Then and now," en, *Program*, vol. 40, no. 3, pp. 219–223, Jul. 2006, ISSN: 0033-0337, 0033-0337. DOI: 10.1108/00330330610681295. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/00330330610681295/full/html (visited on 05/24/2023).

[30] D. Wang, J. Su, and H. Yu, "Feature extraction and analysis of natural language processing for deep learning english language," *IEEE Access*, vol. 8, pp. 46 335–46 345, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2974101.

[31] A. I. Kadhim, "Survey on supervised machine learning techniques for automatic text classification," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 273–292, Jun. 1, 2019, ISSN: 1573-7462. DOI: 10.1007/s10462-018-09677-1. [Online]. Available: https://doi.org/10.1007/s10462-018-09677-1 (visited on 02/13/2023).

[32] C. Santos, M. Nouwens, M. Toth, N. Bielova, and V. Roca, "Consent management platforms under the GDPR: Processors and/or controllers?" In *Privacy Technologies and Policy*, N. Gruschka, L. F. C. Antunes, K. Rannenberg, and P. Drogkaris, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 47–69, ISBN: 978-3-030-76663-4. DOI: 10.1007/978-3-030-76663-4_3.

[33] P. Papadopoulos, P. Snyder, D. Athanasakis, and B. Livshits. "Keeping out the masses: Understanding the popularity and implications of internet paywalls." (2019), [Online]. Available: https://arxiv.org/abs/1903.01406.

[34] D. Bollinger, K. Kubicek, C. Cotrini, and D. Basin, "Automating Cookie Consent and GDPR Violation Detection," en,

[35] M. Hashemi, "Web page classification: A survey of perspectives, gaps, and future directions," *Multimedia Tools and Applications*, vol. 79, no. 17, pp. 11 921–11 945, May 2020, ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-019-08373-8. [Online]. Available: http://link.springer.com/10.1007/s11042-019-08373-8 (visited on 03/28/2023).

[36] H. Alvari, P. Shakarian, and J. E. K. Snyder, "Semi-supervised learning for detecting human trafficking," *Security Informatics*, vol. 6, no. 1, p. 1, Dec. 2017, ISSN: 2190-8532. DOI: 10.1186/s13388-017-0029-8. [Online]. Available: http://security-informatics.springeropen.com/articles/10.1186/s13388-017-0029-8 (visited on 03/28/2023).

[37] E. Hamdy, *Neural models for offensive language detection*, May 30, 2021. DOI: 10.48550/arXiv.2106.14609. arXiv: 2106.14609[cs]. [Online]. Available: http://arxiv.org/abs/2106.14609 (visited on 03/14/2023).

[38] L. Deng, X. Du, and J.-z. Shen, "Web page classification based on heterogeneous features and a combination of multiple classifiers," *Frontiers of Information*

Technology & Electronic Engineering, vol. 21, no. 7, pp. 995–1004, Jul. 1, 2020, ISSN: 2095-9230. DOI: 10.1631/FITEE.1900240. [Online]. Available: https://doi.org/10.1631/FITEE.1900240 (visited on 03/28/2023).

[39] S. M. Mirtaheri, M. E. Dinçktürk, S. Hooshmand, G. V. Bochmann, G.-V. Jourdan, and I. V. Onut, *A brief history of web crawlers*, May 4, 2014. arXiv: 1405.0749[cs]. [Online]. Available: http://arxiv.org/abs/1405.0749 (visited on 03/29/2023).

[40] M. Gray. "Internet growth and statistics: Credits and background," Internet Growth and Statistics: Credits and Background. (), [Online]. Available: https://www.mit.edu/~mkgray/net/background.html (visited on 03/29/2023).

[41] "What is googlebot | google search central | documentation | google developers." (), [Online]. Available: https://developers.google.com/search/docs/crawling-indexing/googlebot (visited on 03/27/2023).

[42] Ansuman Prusty, Pavan Kancherlapalli, Roland Schiebel, Aniket Shah, Aditya Suresh, Oscar Mejia, and Harvard University, "Horizontally scalable web crawler using containerization and a graphical user interface," *International Journal of Engineering Research and*, vol. V9, no. 5, IJERTV9IS050268, May 15, 2020, ISSN: 2278-0181. DOI: 10.17577/IJERTV9IS050268. [Online]. Available: https://www.ijert.org/horizontally-scalable-web-crawler-using-containerization-and-a-graphical-user-interface (visited on 03/27/2023).

[43] D. Eyzenakh, A. Rameykov, and I. Nikiforov, "HIGH PERFORMANCE DISTRIBUTED WEB-SCRAPER," vol. 33, no. 3, pp. 87–100, 2021, ISSN: 2079-8156. [Online]. Available: https://cyberleninka.ru/article/n/high-performance-distributed-web-scraper (visited on 03/27/2023).

[44] J. Cho and H. Garcia-Molina, "Parallel crawlers," in *Proceedings of the 11th international conference on World Wide Web*, Honolulu Hawaii USA: ACM, May 7, 2002, pp. 124–135, ISBN: 978-1-58113-449-0. DOI: 10.1145/511446.511464. [Online]. Available: https://dl.acm.org/doi/10.1145/511446.511464 (visited on 03/29/2023).

[45] R. S. Chaulagain, S. Pandey, S. R. Basnet, and S. Shakya, "Cloud based web scraping for big data applications," in *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, New York, NY: IEEE, Nov. 2017, pp. 138–143, ISBN: 978-1-5386-3684-8. DOI: 10.1109/SmartCloud.2017.28. [Online]. Available: http://ieeexplore.ieee.org/document/8118431/ (visited on 03/29/2023).

[46] S. Kelly, "What is python?" In *Python, PyGame and Raspberry Pi Game Development*. Berkeley, CA: Apress, 2016, pp. 3–5, ISBN: 978-1-4842-2517-2. DOI: 10.1007/978-1-4842-2517-2_2. [Online]. Available: http://link.springer.com/10.1007/978-1-4842-2517-2_2 (visited on 04/27/2023).

[47] Dave Kuhlman. "A python book: Beginning python, advanced python, and python exercises." (Apr. 22, 2012), [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html (visited on 04/27/2023).

[48] P. Krill. "Python popularity still soaring," InfoWorld. (Aug. 8, 2022), [Online]. Available: https://www.infoworld.com/article/3669232/python-popularity-still-soaring.html (visited on 04/27/2023).

[49] F. Yin, X. He, and Z. Liu, "Research on scrapy-based distributed crawler system for crawling semi-structure information at high speed," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018, pp. 1356–1359. DOI: 10.1109/CompComm.2018.8781062.

[50] "Beautiful soup documentation." (), [Online]. Available: https://beautiful-soup-4.readthedocs.io/en/latest/ (visited on 02/21/2023).

[51] *State of Postgres*, en. [Online]. Available: https://www.timescale.com/state-of-postgres/2022 (visited on 05/22/2023).

[52] *What are ACID Transactions?* en-US, Jul. 2021. [Online]. Available: https://www.databricks.com/glossary/acid-transactions (visited on 05/23/2023).

[53] *Docker images for the Selenium Grid Server*, original-date: 2014-11-17T17:39:22Z, May 2023. [Online]. Available: https://github.com/SeleniumHQ/docker-selenium (visited on 05/23/2023).

[54] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2019, ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23386. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_01B-3_LePochat_paper.pdf (visited on 02/10/2023).

[55] *Amazon EKS optimized Amazon Linux AMIs - Amazon EKS*. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-ami.html (visited on 05/28/2023).

[56] *Docker Official Images*, en, May 2023. [Online]. Available: https://docs.docker.com/docker-hub/official_images/ (visited on 05/23/2023).

[57] *Website URL Category Check*, en. [Online]. Available: http://www.cyren.com/security-center/ip-reputation-check (visited on 05/26/2023).

Appendix A: Figures

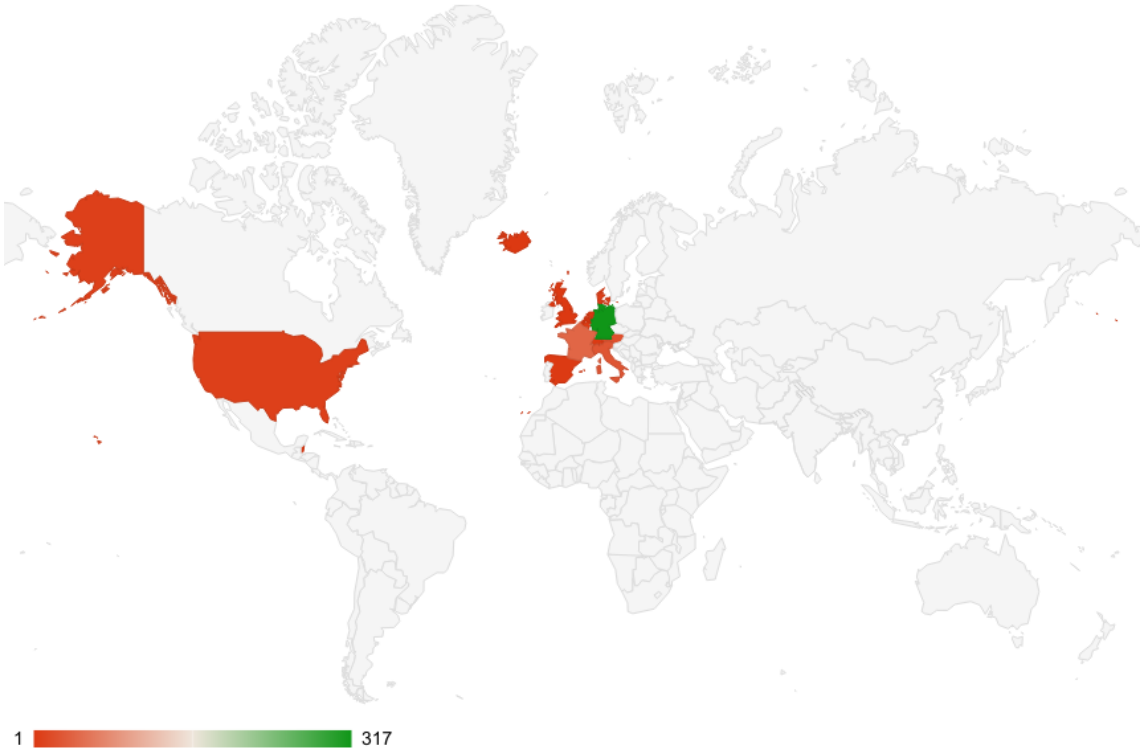


Figure 1: Heatmap of where cookie paywalls were found. As our initial dataset from Morel *et al.* [9] were cookie paywalls from Central Europe, there is likely a strong bias in the feature extraction. There is also the possibility that there are a disproportionate amount of cookie paywall sites based in Europe.

Category Name
Business
Computers & Technology
Education
Entertainment
Fashion & Beauty
File Repository
Finance
Forums & Newsgroups
Games
General
Health & Medicine
Leisure & Recreation
News
Non-profits & NGOs
Personal Sites
Personal Storage
Politics & Law
Real Estate
Religion
Restaurants & Dining
Shopping
Sports
Streaming Media & Downloads
Transportation
Travel
Unknown

Table 1: The different categories that resulted from inputting suspected cookie paywalls into Cyrens URL category checker [57]

Appendix B: Code Snippets

Table 2: List of arguments

Argument	Description
--urls	Comma-separated list of URLs to visit
--file	File containing a list of URLs to visit
--csvfile	CSV-file with the list of URLs to visit. Range in the format: file:10:1000
--firefox	Use Firefox driver
--chrome	Use Chrome driver
--safari	Use Safari driver
--edge	Use Edge driver
--headless	Use headless browser
--grid	Use grid driver
--verbose	Print verbosely
--modals	Search for and process modals
--postgres	Connect to Postgres instead of SQLite
--timer	Record and print time for each process
--client	Fetch loop of URLs from database

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>HTML Example</title>
  </head>
  <body>
    <section>
      <header>
        <h1>This is a section heading.</h1>
      </header>
      <p id="demo">This is a paragraph.</p>
    </section>
  </body>

  <script>
    function myFunction() {
      document.getElementById("demo").innerHTML = "Paragraph changed.";
    }
  </script>
</html>
```

Listing 2: Example HTML tree

Table 3: List of rows in database table

Field Name	Description
url	URL of the site to be crawled, primary key.
rank	Popularity ranking of site.
date_taken_chrome	Timestamp of when URL was claimed.
date_taken_safari	Timestamp of when URL was claimed.
date_taken_msedge	Timestamp of when URL was claimed.
date_taken_firefox	Timestamp of when URL was claimed.
date_finished_chrome	Timestamp when result was received for chrome.
date_finished_safari	Timestamp when result was received for safari.
date_finished_msedge	Timestamp when result was received for msedge.
date_finished_firefox	Timestamp when result was received for firefox.
id_chrome	ID of claimant using chrome.
id_safari	ID of claimant using safari.
id_msedge	ID of claimant using msedge.
id_firefox	ID of claimant using firefox.
result_chrome	Result of the crawl using chrome.
result_safari	Result of the crawl using safari.
result_msedge	Result of the crawl using msedge.
result_firefox	Result of the crawl using firefox.
features_chrome	Feature extraction results using chrome.
features_safari	Feature extraction results using safari.
features_msedge	Feature extraction results using msedge.
features_firefox	Feature extraction results using firefox.
error_chrome	Error type and message using chrome.
error_safari	Error type and message using safari.
error_msedge	Error type and message using msedge.
error_firefox	Error type and message using firefox.

```
FROM docker.io/python:3.10
RUN pip install pipenv
ENV CRAWLER_DIR /crawler
WORKDIR ${CRAWLER_DIR}
COPY Pipfile Pipfile.lock urls.json ${CRAWLER_DIR}/
COPY src ${CRAWLER_DIR}/src
RUN pipenv install --system --deploy
```

Listing 3: Dockerfile used when building the Crawler Container Image

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
  labels:
    app: postgres
data:
  POSTGRES_DB: postgresdb
  POSTGRES_HOST: postgres
  POSTGRES_USER: admin
  POSTGRES_PASSWORD: "spelaringenroll"
  POSTGRES_PORT: "5432"
```

Listing 4: The values from this ConfigMap will be used as environment variables for the Postgres and crawler deployments.

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: postgres-pv-volume
  labels:
    type: local
    app: postgres
spec:
  storageClassName: manual
  capacity:
    storage: 75Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pv-claim
  labels:
    app: postgres
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 75Gi
```

Listing 5: This configuration file declares the persistent data storage and claim for the Postgres database.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:15.2
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 5432
          envFrom:
            - configMapRef:
                name: postgres-config
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgredb
      volumes:
        - name: postgredb
          persistentVolumeClaim:
            claimName: postgres-pv-claim
```

Listing 6: Postgres Kubernetes deployment.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres
  labels:
    app: postgres
spec:
  type: NodePort
  ports:
   - port: 5432
  selector:
   app: postgres
```

Listing 7: A Postgres Service configuration, which exposes port 5432 to the pods in the namespace.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: selenium-hub
  labels:
    app: selenium
spec:
  replicas: 1
  selector:
    matchLabels:
      app: selenium
  template:
    metadata:
      labels:
        app: selenium
        role: hub
    spec:
      restartPolicy: Always
      containers:
      - name: hub
        image: selenium/hub:4.9.1-20230508
        ports:
        - name: http
          containerPort: 4444
        resources:
          requests:
            cpu: 4
            memory: 8G
          limits:
            cpu: 8
            memory: 16G
```

Listing 8: This YAML-file is the Kubernetes configuration for the Selenium Hub deployment.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: selenium-hub
spec:
  ports:
    - name: hub
      port: 4444
      targetPort: http
    - name: subscribe
      port: 4443
      targetPort: 4443
    - name: publish
      port: 4442
      targetPort: 4442
  selector:
    app: selenium
    role: hub
```

Listing 9: This file shows the Service exposing the Selenium Hub to the rest of the namespace.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: selenium-firefox
  labels:
    app: selenium
spec:
  replicas: 16
  selector:
    matchLabels:
      app: selenium
  template:
    metadata:
      labels:
        app: selenium
        role: firefox
    spec:
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      restartPolicy: Always
      containers:
      - name: firefox
        image: selenium/node-firefox:4.9.0-20230421
        env:
        - name: SE_EVENT_BUS_HOST
          value: "selenium-hub"
        - name: SE_EVENT_BUS_PUBLISH_PORT
          value: "4442"
        - name: SE_EVENT_BUS_SUBSCRIBE_PORT
          value: "4443"
        - name: SE_NODE_SESSION_TIMEOUT
          value: "30"
        ports:
        - name: http
          containerPort: 5555
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        resources:
          requests:
            cpu: ".5"
            memory: 1G
          limits:
            cpu: "2"
            memory: 2G
```

Listing 10: This YAML-configuration file is the Kubernetes configuration for a set of 16 Selenium Firefox browsers.

Appendix C: Cookie Paywalls

The table below lists all 431 websites on which we found cookie paywalls.

URLs	Category	Country	Price/month
0180.info	Business	Germany	€2.99
24o.it	News	Italy	€1.00
50plus.de	Shopping	Germany	€2.99
750g.com	Restaurants & Dining	France	€2.00
abitare.it	Business	Italy	€0.75
additive-manufacturing-industry.de	Business	Germany	€2.99
aerokurier.de	Transportation	Germany	€2.99
aerotelegraph.com	Travel	USA	€3.30
aerztezeitung.de	Health & Medicine	Germany	€2.99
allesbeste.de	Education	Germany	€2.99
allgemeine-zeitung.de	News	Germany	€2.99
allocine.fr	Streaming Media & Downloads	France	€2.00
amandinecooking.com	Restaurants & Dining	France	€2.00
amica.it	Fashion & Beauty	Italy	€0.75
analog-praxis.de	Shopping	Germany	€2.99
androidnext.de	Computers & Technology	Germany	€1.99
anime2you.de	Entertainment	Germany	€2.99
ansa.it	News	Italy	€1.42
ansamed.info	News	Italy	€1.42
astroportal.com	Religion	Austria	€4.16
atlantico.fr	News	France	€9.90
aufunsere.art	Entertainment	Austria	€2.99
auszeit.bio	Personal Sites	Germany	€2.99
autlook.at	Business	Austria	€2.99
auto-motor-und-sport.de	Transportation	Germany	€0.99
autobild.de	Transportation	Germany	€3.99
autoflotte.de	Transportation	Germany	€3.99
autofrage.net	Transportation	Germany	€2.99
autoguru.de	Transportation	Germany	€2.99
autohaus.de	Transportation	Germany	€2.99
automobil-industrie.vogel.de	Business	Germany	€2.99
autoservice.co.at	Transportation	Austria	€2.99
autoservicepraxis.de	Transportation	Germany	€2.99
backenmachtgluecklich.de	Restaurants & Dining	Germany	€2.99
bahnblogstelle.com	Unknown	Germany	€2.99
basic-tutorials.de	Computers & Technology	Germany	€2.99
baugewerbe-magazin.de	News	Germany	€2.99
bba-online.de	Computers & Technology	Germany	€2.99
beauxarts.com	General	France	€3.95
bento.de	Entertainment	Germany	€4.99
bigdata-insider.de	Business	Germany	€2.99
bike-x.de	Transportation	Germany	€2.99
bild.de	News	Germany	€3.99
blechnet.com	Business	Germany	€2.99
boersennews.de	Finance	Germany	€1.99
boomer.at	Unknown	Austria	€2.99
braunschweiger-zeitung.de	News	Germany	€2.99
bravo.de	Entertainment	Germany	€2.99
brigitte.de	Leisure & Recreation	Germany	€4.99
bz-berlin.de	News	Germany	€3.99
calcionapoli1926.it	Sports	Italy	€0.83
caminteresse.fr	News	France	€1.99
capital.de	Finance	Germany	€4.99
capital.fr	News	France	€1.99
caravaning.de	Travel	Germany	€2.99
cavallo.de	Business	Germany	€2.99
cesoirtv.com	Entertainment	France	€1.99
chartsinfrance.net	Entertainment	France	€1.00
cinefacts.de	Entertainment	Germany	€1.99
cittaceleste.it	Sports	Italy	€0.83
climate-data.org	News	USA	€2.99
cloudcomputing-insider.de	Computers & Technology	Germany	€2.99
cnv-medien.de	News	Germany	€1.99
computer-automation.de	Computers & Technology	Germany	€2.99

computerbase.de	Computers & Technology	Germany	€4.00
computerbild.de	Computers & Technology	Germany	€2.99
computerfrage.net	Computers & Technology	Germany	€2.99
connect-living.de	Shopping	Germany	€2.99
connect-professional.de	Unknown	Germany	€2.99
connect.de	Computers & Technology	Germany	€2.99
corriere.it	News	Italy	€1.25
cosmopolitan.de	Leisure & Recreation	Germany	€2.99
das-pta-magazin.de	Health & Medicine	Germany	€2.99
datacenter-insider.de	Computers & Technology	Germany	€2.99
derbyderbyderby.it	Sports	Italy	€0.83
derstandard.at	News	Austria	€8.90
derstandard.de	News	Germany	€8.90
derwesten.de	News	Germany	€2.99
desired.de	Shopping	Germany	€1.99
dev-insider.de	Computers & Technology	Germany	€2.99
devicemed.de	Business	Germany	€2.99
dieharke.de	News	Germany	€4.99
diestandard.at	News	Austria	€8.90
dino-online.de	Forums & Newsgroups	Germany	€2.99
dnn-online.de	News	Germany	€4.99
dnn.de	News	Germany	€4.99
donnerwetter.de	News	Germany	€2.99
dumontreise.de	Travel	Germany	€2.99
easyvoyage.com	Travel	France	€2.00
eat.de	Computers & Technology	Germany	€2.99
echo-online.de	News	Germany	€2.99
egovernment-computing.de	Computers & Technology	Germany	€2.99
eklablog.com	Computers & Technology	France	€1.00
ekladata.com	Computers & Technology	France	€1.00
elektroniknet.de	Business	Germany	€2.99
elektronikpraxis.de	News	Germany	€2.99
elektropraxis.at	Business	Austria	€2.99
elektrotechnik.vogel.de	Business	Germany	€2.99
eltern.de	Leisure & Recreation	Germany	€4.99
embedded-software-engineering.de	Computers & Technology	Germany	€2.99
emtb-news.de	News	Germany	€2.99
en.kunststoffe.de	Business	Germany	€2.99
entsorgen.org	Computers & Technology	Switzerland	€2.99
erbrecht-ratgeber.de	Business	Germany	€2.99
erdbeerlounge.de	Leisure & Recreation	Germany	€1.99
erneuerbareenergien.de	Business	Germany	€4.99
esports.com	Sports	Germany	€2.99
etmm-online.com	Entertainment	Germany	€2.99
eurogamer.de	Games	Germany	€2.50
europamester.dk	Sports	Denmark	€3.89
express.de	News	Germany	€4.99
factorynet.at	Business	Austria	€2.99
fahrschule-online.de	Transportation	Germany	€2.99
falk.de	Education	Germany	€2.99
familie.de	Education	Germany	€1.99
farmeramania.de	Games	Germany	€1.04
faszination-fankurve.de	Sports	Germany	€2.99
faz.de	News	Germany	€4.99
faz.net	News	Germany	€4.99
fcinter1908.it	Sports	Italy	€0.83
femmeactuelle.fr	Leisure & Recreation	France	€1.99
finance-magazin.de	Finance	Germany	€5.90
finanzfrage.net	Finance	Germany	€2.99
finanzmarktwelt.de	News	Germany	€49.00
firmenwagen.co.at	Computers & Technology	Austria	€2.99
flugrevue.de	Travel	Germany	€2.99
fn.de	News	Germany	€2.99
form-werkzeug.de	Business	Germany	€2.99
formel1.de	Sports	Germany	€2.99
forzaroma.info	News	Italy	€0.83
fraenkische-rezepte.de	Restaurants & Dining	Germany	€2.99
free-fonts.com	Computers & Technology	USA	€2.99
freenet.de	Search Engines & Portals	Germany	€2.99
funandnews.de	News	Germany	€2.99

funkschau.de	Computers & Technology	Germany	€2.99
fussballfieber.de	Shopping	Germany	€2.99
gabler-banklexikon.de	Education	Germany	€2.99
gala.de	Fashion & Beauty	Germany	€4.99
gala.fr	Entertainment	France	€1.99
gamestar.de	Games	Germany	€4.99
gamona.de	Games	Germany	€1.99
gartendialog.de	Computers & Technology	Germany	€2.99
gartenjournal.net	Personal Sites	Germany	€2.99
gartenlexikon.de	Health & Medicine	Germany	€2.99
gastromand.dk	Restaurants & Dining	Denmark	€3.89
gazzetta.it	News	Italy	€0.83
geb-info.de	Business	Germany	€4.99
gentside.com	Leisure & Recreation	France	€1.99
geo.de	Leisure & Recreation	Germany	€4.99
geo.fr	Leisure & Recreation	France	€1.99
gesund-verlieben.de	Business	Germany	€2.99
gesundheitsfrage.net	Health & Medicine	Germany	€2.99
gifhorner-rundschau.de	News	Germany	€2.99
giga.de	Computers & Technology	Germany	€1.99
glaswelt.de	Business	Germany	€4.99
gnz.de	News	Germany	€4.99
goettinger-tageblatt.de	News	Germany	€4.99
goquiz.dk	Entertainment	Denmark	€3.89
goslarsche.de	News	Germany	€2.99
gpsradler.de	Computers & Technology	Germany	€2.50
gutefrage.de	Computers & Technology	Germany	€2.99
gutefrage.net	Forums & Newsgroups	Germany	€2.99
gutekueche.at	Search Engines & Portals	Austria	€2.99
gutekueche.de	Restaurants & Dining	Germany	€2.99
hallo-eltern.de	Health & Medicine	Germany	€2.99
hanser-automotive.de	Business	Germany	€2.99
harvardbusinessmanager.de	Business	Germany	€4.99
harzkurier.de	News	Germany	€2.99
haus-garten-test.de	Computers & Technology	Germany	€2.99
hausgarten.net	Leisure & Recreation	Germany	€2.99
haz.de	News	Germany	€4.99
hbold.dk	Sports	Denmark	€3.89
healthcare-computing.de	Health & Medicine	Germany	€2.99
heilpraxisnet.de	Health & Medicine	Germany	€2.99
heise.de	News	Germany	€4.95
helmstedter-nachrichten.de	News	Germany	€2.99
hildesheimer-allgemeine.de	News	Germany	€2.99
hlk.co.at	Business	Austria	€2.99
hockeymagasinet.dk	Sports	Denmark	€3.89
hoerzu.de	Entertainment	Germany	€2.99
hortica.de	Unknown	Germany	€2.99
huffingtonpost.fr	News	France	€2.00
huffingtonpost.it	News	Italy	€1.00
huffpost.fr	News	France	€2.00
iban-rechner.de	Finance	Germany	€2.99
ibancalculator.com	Finance	Iceland	€2.99
ilclubdellericette.it	Leisure & Recreation	Italy	unknown
ilmilanista.it	Sports	Italy	€0.83
ilposticipo.it	Sports	France	€0.83
ilsecoloxix.it	News	Italy	€1.00
ilsole24ore.com	News	Italy	€1.00
impulse.de	Business	Germany	€5.90
industrial-production.de	News	Germany	€2.99
industriemagazin.at	News	Austria	€2.99
industry-of-things.de	Business	Germany	€2.99
infranken.de	News	Germany	€2.99
inrlp.de	Personal Sites	Germany	€2.99
inside-digital.de	Computers & Technology	Germany	€2.99
iodonna.it	News	Italy	€0.75
ip-insider.de	Computers & Technology	Germany	€2.99
it-business.de	Computers & Technology	Germany	€2.99
it-daily.net	Computers & Technology	Germany	€2.99
itasportpress.it	Sports	Italy	€0.83
jeuxvideo.com	Forums & Newsgroups	France	€2.00

juvenews.eu	Sports	Belgium	€0.83
karrierefragen.de	Education	Germany	€2.99
kicker.de	News	Germany	€2.49
kino.de	Entertainment	Germany	€1.99
klack.de	Entertainment	Germany	€2.99
kn-online.de	News	Germany	€4.99
korrekturen.de	Education	Germany	€2.99
krone.at	News	Austria	€11.90
ksta.de	News	Germany	€4.99
kunststoff-magazin.de	Business	Germany	€2.99
kunststoffe.de	Business	Germany	€2.99
kurier.at	News	Austria	€3.60
lab-worldwide.com	Business	Germany	€2.99
labo.de	Business	Germany	€2.99
laborpraxis.vogel.de	Business	Germany	€2.99
lachainemeteo.com	News	France	€2.29
landeszeitung.de	News	Germany	€4.99
lanline.de	Computers & Technology	Germany	€2.99
lastampa.it	News	Italy	€1.00
le10sport.com	Sports	France	€2.00
leblogtvnews.com	Personal Sites	France	€2.00
lecker.de	Restaurants & Dining	Germany	€2.99
leiweb.it	Entertainment	Italy	€0.75
lemonde.fr	News	France	€10.99
lepoint.fr	News	France	€1.00
lezappingdupaf.com	Entertainment	France	€2.00
liberation.com	News	France	€1.00
liberation.fr	News	France	€1.00
liebenswert-magazin.de	Health & Medicine	Germany	€2.99
likehifi.de	Computers & Technology	Germany	€2.99
linux-community.de	Forums & Newsgroups	Germany	€2.99
linux-magazin.de	Computers & Technology	Germany	€2.99
linux-user.de	News	Germany	€2.99
livingathome.de	Shopping	Germany	€4.99
ln-online.de	News	Germany	€4.99
logistik-heute.de	Business	Germany	€2.99
logistra.de	Shopping	Germany	€2.99
lonelyplanet.de	Travel	Germany	€2.99
lvz-online.de	News	Germany	€4.99
lvz.de	News	Germany	€4.99
macnews.de	Computers & Technology	Germany	€1.99
macwelt.de	Computers & Technology	Germany	€2.99
maennersache.de	Entertainment	Germany	€2.99
maennerseite.net	Business	Germany	€2.99
maerkischeallgemeine.de	News	Germany	€4.99
manager-magazin.de	News	Germany	€4.99
marconomy.de	Business	Germany	€2.99
marcopolo.de	Travel	Germany	€2.99
maschinenmarkt.ch	Business	Germany	€2.99
maschinenmarkt.international	Business	Germany	€2.99
maschinenmarkt.vogel.de	Business	Germany	€2.99
materialfluss.de	Business	Germany	€2.99
maz-online.de	News	Germany	€4.99
medical-design.news	Real Estate	Germany	€2.99
meineorte.com	Leisure & Recreation	USA	€2.99
menshealth.de	Health & Medicine	Germany	€2.99
meteoblue.com	News	Switzerland	€0.75
millenium.org	Forums & Newsgroups	France	€1.00
mission-additive.de	Business	Germany	€2.99
mittelhessen.de	News	Germany	€2.99
mm-logistik.vogel.de	Business	Germany	€2.99
moin.de	Computers & Technology	Germany	€2.99
mondoudinese.it	News	Italy	€9.99
motorradfrage.net	Forums & Newsgroups	Germany	€2.99
motorradonline.de	News	Germany	€2.99
motorradundreisen.de	Transportation	Germany	€2.19
motorsport-total.com	Sports	Germany	€2.99
mountainbike-magazin.de	Sports	Germany	€2.99
moviemaze.de	Entertainment	Germany	€1.99
mtb-news.de	Sports	Germany	€2.99

mummum.dk	Personal Sites	Denmark	€2.55
mz-web.de	News	Germany	€1.99
neon.de	Business	Germany	€4.99
neonmag.fr	Entertainment	France	€1.99
netzwelt.de	Computers & Technology	Germany	€2.99
neuepresse.de	News	Germany	€2.99
news38.de	News	Germany	€2.99
next-mobility.de	Business	Germany	€2.99
nn-online.de	News	Germany	€2.90
nn.de	Education	Germany	€2.90
nordbayern.de	News	Germany	€2.90
nz-online.de	News	Germany	€2.90
oberhessische-zeitung.de	News	Germany	€2.99
oekotest.de	Leisure & Recreation	Germany	€2.99
ohmymag.com	Leisure & Recreation	Germany	€1.99
omnibusrevue.de	Business	Germany	€2.99
op-marburg.de	News	Germany	€4.99
ostsee-zeitung.de	News	Germany	€4.99
over-blog-kiwi.com	Personal Sites	France	€2.00
over-blog.com	Personal Sites	France	€2.00
overblog.com	Personal Sites	France	€2.00
paz-online.de	News	Germany	€4.99
pc-magazin.de	Computers & Technology	Germany	€2.99
pcgames.de	Entertainment	Germany	€2.50
pcwelt.de	Computers & Technology	Germany	€2.99
peiner-nachrichten.de	Politics & Law	Germany	€2.99
personalwirtschaft.de	Shopping	Germany	€5.90
photovoltaik.eu	Business	Belgium	€4.99
pianetamilan.it	Sports	Italy	€9.99
plantopedia.de	Education	Germany	€2.99
pnn.de	News	Germany	€2.99
prad.de	Computers & Technology	Germany	€2.99
praxisvita.de	Health & Medicine	Germany	€2.99
prisma.de	Computers & Technology	Germany	€2.99
process-worldwide.com	Non-profits & NGOs	Germany	€2.99
process.vogel.de	Health & Medicine	Germany	€2.99
profi-werkstatt.net	News	Germany	€2.99
programme-tv.net	Streaming Media & Downloads	France	€1.99
programme.tv	News	USA	€1.99
promiflash.de	Entertainment	Germany	€2.00
promobil.de	General	Germany	€2.99
purebreak.com	Entertainment	France	€1.00
purepeople.com	Leisure & Recreation	France	€2.00
puretrend.com	Entertainment	France	€1.00
qz-online.de	Business	Germany	€2.99
radiobielefeld.de	Entertainment	Germany	€2.99
radioguetersloh.de	Entertainment	Germany	€2.99
radiohochstift.de	Entertainment	Germany	€2.99
radiolippe.de	Streaming Media & Downloads	Germany	€2.99
radsport-news.com	Sports	Germany	€2.99
raspberry-pi-geek.de	Business	Germany	€2.99
reisefrage.net	Travel	Germany	€2.99
rennrad-news.de	Sports	Germany	€2.99
repubblica.it	News	Italy	€1.00
rheinpfalz.de	News	Germany	€1.50
rnd.de	News	Germany	€4.99
roadbike.de	Sports	Germany	€2.99
rp-online.de	News	Germany	€4.00
runnersworld.de	Sports	Germany	€2.99
saechsische.de	News	Germany	€2.99
sbz-monteur.de	Education	Germany	€4.99
sbz-online.de	Business	Germany	€4.99
schifffahrtundtechnik.de	Business	Germany	€2.99
schoener-wohnen.de	Shopping	Germany	€4.99
schulferien.org	Computers & Technology	Germany	€2.99
sciencesetavenir.fr	News	France	€4.00
security-insider.de	Computers & Technology	Germany	€2.99
selbst.de	Personal Sites	Germany	€2.99
selfies.com	Entertainment	Great Britain	€2.99
siegener-zeitung.de	News	Germany	€4.99

smart-rechner.de	Computers & Technology	Germany	€2.99
smarterworld.de	Business	Germany	€2.99
smarthouse-pro.de	Personal Sites	Germany	€2.99
sn-online.de	News	Germany	€4.99
solidbau.at	Business	Austria	€2.99
spiegel.de	News	Germany	€4.99
spiegel.tv	General	Germany	€4.99
spielaffe.de	Games	Germany	€1.99
spieletipps.de	Games	Germany	€1.99
spielfilm.de	Entertainment	Germany	€2.99
spon.de	News	Germany	€4.99
sportlerfrage.net	Sports	Germany	€2.99
sportnews.bz	Sports	Belize	€9.99
springermedizin.at	Health & Medicine	Austria	€2.99
springermedizin.de	Health & Medicine	Germany	€2.99
springerpflege.de	Health & Medicine	Germany	€2.99
springerprofessional.de	Business	Germany	€2.99
sprit-plus.de	Computers & Technology	Germany	€2.99
stadionwelt.de	Sports	Germany	€2.99
stern.de	News	Germany	€4.99
stimme.de	News	Germany	€2.99
stol.it	News	Italy	€9.99
storage-insider.de	Computers & Technology	Germany	€2.99
stylevamp.de	Business	Germany	€2.99
sudoku-aktuell.de	Leisure & Recreation	Germany	€2.99
sudoku-topical.com	Business	USA	€2.99
sudouest.fr	News	France	€9.90
supermadame.org	Personal Sites	France	€2.00
swp.de	News	Germany	€9.92
sz-online.de	News	Germany	€2.99
t-online.de	Search Engines & Portals	Germany	€2.99
t3n.de	Computers & Technology	Germany	€1.90
tageblatt.de	News	Germany	€2.99
tagesspiegel.de	News	Germany	€2.99
techniker-forum.de	Forums & Newsgroups	Germany	€2.00
teltarif.de	Computers & Technology	Germany	€2.99
terrafemina.com	News	France	€1.00
teslamag.de	Transportation	Germany	€2.99
tga.at	Business	Austria	€2.99
thueringen24.de	News	Germany	€2.99
tierfans.net	Shopping	Germany	€2.99
tomaten.de	Shopping	Germany	€2.99
tonline.de	Search Engines & Portals	Germany	€2.99
traktuell.at	Computers & Technology	Austria	€2.99
transferxl.com	Personal Storage	Netherlands	€8.46
trucker.de	Forums & Newsgroups	Germany	€2.99
tuttobolognaweb.it	Sports	Italy	€9.99
tvdigital.de	Entertainment	Germany	€2.99
tvdirekt.de	Entertainment	Germany	€2.99
tvmovie.de	Entertainment	Germany	€2.99
unnuetzes.com	Entertainment	Germany	€2.99
unsere-helden.com	Computers & Technology	USA	€2.99
unterwegs-auf-der-autobahn.de	Computers & Technology	Germany	€2.99
urbia.de	Health & Medicine	Germany	€4.99
utopia.de	Leisure & Recreation	Germany	€2.99
verkehrsrundschau.de	News	Germany	€2.99
versicherungsmagazin.de	Finance	Germany	€2.99
vienna.at	News	Austria	€4.80
violanews.com	Sports	Italy	€9.99
vision-mobility.de	Computers & Technology	Germany	€2.99
voici.fr	Entertainment	France	€1.99
vol.at	News	Austria	€4.80
volksstimme.de	News	Germany	€1.99
vorname.com	Leisure & Recreation	Germany	€4.99
wallstreet-online.de	Finance	Germany	€1.99
watson.de	News	Germany	€2.99
waz-online.de	News	Germany	€4.99
we-go-wild.com	General	Austria	€2.99
weihnachtsmarkt-deutschland.de	Business	Germany	€2.99
welt.de	News	Germany	€1.99

werkstatt-betrieb.de	Business	Germany	€2.99
werstreamt.es	Entertainment	Spain	€2.99
wetter.com	News	Germany	€2.99
wetter.info	News	Germany	€2.99
wiesbadener-kurier.de	Search Engines & Portals	Germany	€2.99
winfuture.de	Computers & Technology	Germany	€2.99
winload.de	File Repository	Germany	€1.99
wir-in-der-praxis.de	Computers & Technology	Germany	€2.99
wirtschaftslexikon.gabler.de	Business	Germany	€2.99
wohnmobilforum.de	Sports	Germany	€2.17
womenshealth.de	Health & Medicine	Germany	€2.99
wormser-zeitung.de	Search Engines & Portals	Germany	€2.99
wunderbunt.de	Leisure & Recreation	Germany	€2.99
wunderweib.de	Leisure & Recreation	Germany	€2.99
ze.tt	News	Spain	€4.80
zeit.de	News	Germany	€4.80
zuhause.de	Business	Germany	€2.99

Table 4: All cookie paywalls found during this thesis, along with categories as reported by Cyren, and prices in terms of euros/month.