ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Generalizing Browser Extension CookieAudit for Auditing Consent Popups' GDPR Compliance

Bachelor's Thesis

Szymon Tomasz Nastaly

August 26, 2024

Advisors: Prof. Dr. D. Basin, Dr. K. Kubicek, A. Bouhoula
Department of Computer Science, ETH Zürich

Abstract

This thesis presents a new version of CookieAudit, a browser extension designed to analyze websites for compliance with the General Data Protection Regulation (GDPR) and ePrivacy Directive (ePD) concerning cookie usage and consent management. As internet users spend an average of more than two hours per day online and are tracked by an average of 17 trackers per website [1], the need for effective privacy protection and regulation compliance has become increasingly important. The new CookieAudit builds upon and extends previous work in the field of automated GDPR compliance checking. The extension generalizes the previous version by Kubicek and Zanga et al. [2]. CookieAudit incorporates machine learning models for cookie classification [3] and natural language processing models for analyzing cookie notice content [4]. The extension provides a semi-automated approach to website auditing. This includes automatic analysis of the cookie notice text and interactive elements, detection of non-essential cookies set without proper consent, identification of dark patterns in cookie notices, and the generation of both PDF and machine-readable JSON reports. CookieAudit aims to overcome the limitations of previous tools by offering in-depth analysis applicable to a wide variety of websites, regardless of the specific *Consent Management Platform* (*CMP*) used. Those features make CookieAudit a valuable tool for website operators, users, and regulators for verifying GDPR compliance.

Contents

Chapter 1

Introduction

The internet has become a central point of personal data collection [1]. Users are spending long periods of time on the internet, with the average European being more than two hours per day [1] online. Furthermore, they are being widely tracked while doing so, with an average of 17 trackers per website.[1]). This has led to the introduction of regulations such as the General Data Protection Regulation (*GDPR*). Enacted in May 2018, it provides a comprehensive privacy legislation across the European Union (*EU*). Expanding on previous regulations (such as the ePrivacy Directive), it has defined additional restrictions for data collection, responsibilities for website owners and rights for users [1].

Websites need to inform users about data collection and its purposes. Additionally, a legal basis such as user consent is needed for non-functional cookies which, e.g., may be used for user tracking. Such regulation is necessary as more than 90% of websites use cookies capable of user identification. In the context of browser cookies, cookie notices are used to both declare data collection and its purposes, but also to obtain user consent. [4]. Bouhoula et al. conducted a comprehensive, automatic analysis of GDPR cookie compliance of 97k popular websites. Despite the potential fines [5], violations are very common according to Bouhoula et al., with 65% of websites ignoring user rejection of cookies [4]. An overview of large-scale studies regarding the adherence to GDPR rules was compiled by Bouhoula et al. [4].

It can be challenging for web page operators, users, and regulators to verify compliance of specific websites with these regulations. Cookie notices often have different designs and implementation details. This complicates automatic verification of whether a notice defines the cookie purposes and offers possibilities to configure or reject non-necessary cookies. Possible approaches include heuristics, manual annotation and machine learning models [1, 4]. Similarly, recognizing what cookies are used for is not trivial [5, 3]. Those issues become apparent in previous large-scale studies: Nouwens et al. [6]

restrict their study to websites implementing five popular Consent Management Platforms (*CMP*). After collecting data from 680 websites from the United Kingdom, they found that only 11.80% of the cookie notices met basic GDPR requirements (namely, no pre-ticked optional boxes, simple way to reject and explicit consent). Matte et al. [7] focus on cookie notices adhering to the IAB TCF specification. They describe violations such as "Implicit consent prior to interaction" in 10% of the websites.

Similarly, tools designed for users to audit specific websites are either fully manual and labor-intensive [8], can only automatically analyze cookie notices by specific CMPs, require IAB TCF compliance [7], or only provide rudimentary analysis [9].

Our Work

In our work, we address the limitations of previous auditing tools. We develop a browser extension based on the work of Bouhoula et al. that interacts with a specific web page to audit its cookie compliance with the GDPR, requiring only minimal user involvement.

Chapter 2

Technical and legal background

This chapter presents the technical foundations of browser extension development and the relevant European Union laws for compliance auditing. The technical background is only a general overview, while the detailed technical implementation of CookieAudit is discussed in Chapter 5.

2.1 Browser extensions

Browser extensions are programs that can extend or change the functionality of browsers. They are built using web technologies (JavaScript, HTML, CSS) and can use the APIs provided by the browser. The APIs of Chromium-based browsers (e.g., Chrome and Edge) overlap in large parts with the APIs of Mozilla's Firefox, while Apple's Safari sometimes differs significantly. As a result, many extensions can run in several browser environments.

When building browser extensions for Chromium-based browsers, developers have to split up their code according to the specification by Manifest Version 3 [10]. In this section, we first explain the different parts of a general browser extension architecture and then detail the challenges posed by Manifest V3. Fig. 2.1 shows a visualization of the building blocks and their interactions.

2.1.1 Extension service workers

The Extension Service Worker is the central location for business logic. By having access to the full WebExtensions API, it is the place to define event listeners,[1] send and receive messages to and from the content scripts and popup or run time intensive computations [11, Ch. 4]. Service workers operate

[1] By registering event listeners at the beginning of the service worker, developers can define code that should handle events. The most relevant events are browser events (e.g., first installation of an extension), tab events (e.g., user navigating to a different URL), and extension events (e.g., messaging between extension components).

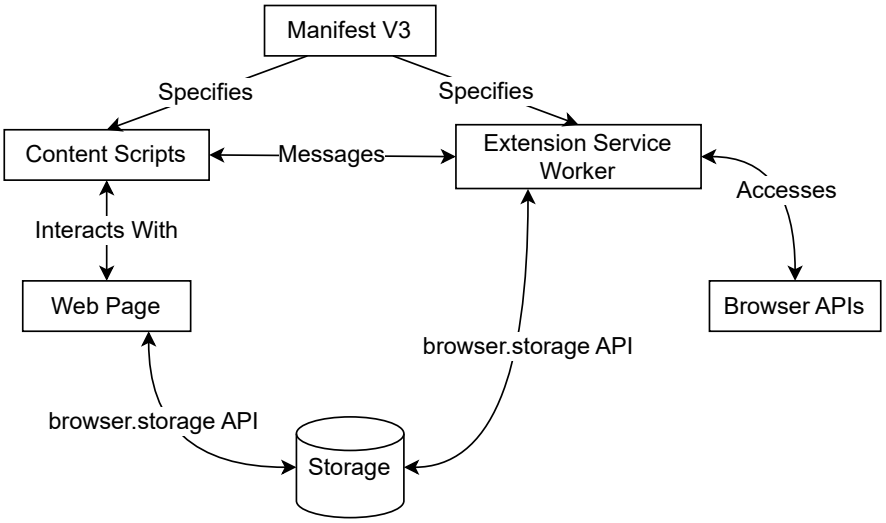


Figure 2.1: The general architecture of a manifest version 3 browser extension.

asynchronously on a separate thread, preventing interference with website execution. They are ephemeral, started by event handlers and terminated when idle. Therefore, developers must persist important application state to storage (see Section 2.1.4) rather than relying on in-memory variables, as these are reset upon termination.

2.1.2 Content scripts

Content scripts are JavaScript programs injected into web pages. They can access and modify the page's HTML structure and CSS styles. By default, content scripts run in an isolated environment, separate from the page's own JavaScript.[2] This isolation prevents direct access between the content script and page script. Variables are not shared, and functions from one context cannot be called by the other. Content scripts have limited access to the WebExtensions API, for example, they cannot change the current tab's URL. To interact with the rest of the extension, content scripts use messaging and a shared key-value store.

2.1.3 Communication

Since the extension service worker and the different content scripts run in separate JavaScript environments, variables and functions cannot be accessed across those boundaries. Instead, browsers provide the messaging API to

[2]Developers can deactivate this isolation by setting the execution world to `MAIN`. Then, the content script will be able to, e.g., read variables of the web page JavaScript.

```
let response = await chrome.runtime.sendMessage({
  cookieNoticeDetected: true
});
```

(a) Sending a message from the content script

```
function handleMsg(request,sender,sendResponse) {
  // ... handle message inside request object ...
  // send response to content script:
  sendResponse({ storedData: true });
}
chrome.runtime.onMessage.addListener(handleMsg);
```

(b) Handling a message in the background script

Figure 2.2: Message passing in Chrome extension

send and listen for messages. For example, in the context of cookie notice detection, if a content script has detected a cookie notice and needs to inform the extension service worker about it, one could send a corresponding message as in Fig. 2.2.

2.1.4 Storage

Browser extensions often need to be able to store data across browser restarts. Developers can therefore use the asynchronous[3] storage API to store key-value pairs serialized as JSON objects. Serializable types include primitives, arrays, and objects[4], but not functions or DOM elements (`HTMLElement`).

2.1.5 Popup

The primary way for users to interact with a browser extension is through a small dialog called *popup*. It can only be opened by the user, that is, by clicking the extension's action icon in the browser toolbar. Popups are programmed using the same web technologies as websites: HTML, CSS, and JavaScript. They have access to all WebExtensions APIs provided by the browser, e.g., storage and messaging. Fig. 2.3 shows the popup of CookieAudit.

2.1.6 Manifest version 3

We developed CookieAudit according to Manifest version 3 [10]. It is the newest specification of the extensions API for Chromium-based browsers.

[3]Even though JavaScript is single-threaded, its asynchronous mechanisms allow non-blocking execution, handling tasks like I/O operations without halting the main thread.

[4]Associative arrays, also known as maps or dictionaries, are called objects in JavaScript.

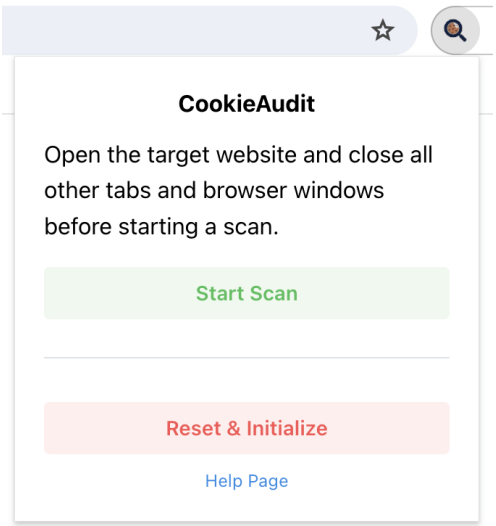


Figure 2.3: Popup dialog of CookieAudit

Compared to Manifest version 2, it includes API changes and restrictions aimed to increase the security and performance of extensions. Version 3 has forced some developers to make considerable changes to their extensions [12, 13], sometimes even reducing the functionality [14]. We present some notable changes between Manifest versions 2 and 3.

From background scripts to service workers

In Manifest version 2, developers could use `background scripts` which effectively were non-visible, persistent web pages. Many extensions developed for manifest version 2 used the fact that background scripts are never terminated and that their global variables are accessible from the extension popup. As explained in Section 2.1.1, developers cannot rely on those characteristics anymore. The migration to manifest version 3 sometimes required considerable changes in the architecture of extensions.[5] Furthermore, the mechanism requiring idle service workers to restart after being shut down can lead to delays in event handling. This time to revival can be problematic for time sensitive extensions.

Execution of arbitrary strings

The execution of arbitrary strings with the `eval()` function could be used for security exploits, as shown by Carlini et al. [16]. To solve this issue, it is no longer possible to evaluate and execute strings using methods such

[5]For example, Gonzalez reported increased complexity following the move of the Bitwarden password manager to Manifest V3 [15]. The switch to service workers made changes in the extension architecture necessary.

as `eval()`, or `new Function()`. It is still possible to evaluate and execute WebAssembly,[6] by specifying `wasm-unsafe-eval` in the extension manifest. The ability to run WebAssembly is necessary for many performance critical applications, such as the client-side inference of AI models.

Other

Manifest version 3 introduces several other significant changes to browser extension development. It limits the inspection and modification of live network requests and prohibits the execution of remotely hosted code.[7] Furthermore, the new version renames and restructures some WebExtensions API functions.

2.2 Cookies

HTTP Cookies are small pieces of text data stored in a user's browser when visiting websites. They are initially created by web servers, sent to clients, and stored by browsers. Each cookie is associated with a specific domain. *First-party* cookies are associated with the domain of the website being visited. For example, when visiting `news.example`, a cookie may be set with that same domain. *Third-party* cookies are associated with domains different from the website being visited. For instance, if `news.example` displays an advertisement served from `ads.example`, the ad server can set cookies with the `ads.example` domain. Cookies are automatically sent back to their originating servers with every HTTP request to that domain. This adds state to the otherwise stateless nature of HTTP.

Cookies serve various purposes. They can provide essential functionality, such as storing user authentication data. However, they can also be used to track users across different websites for analytics or advertising purposes. This is done by creating a unique, identifying key for each user and storing it as a cookie.

2.3 General Data Protection Regulation and ePrivacy Directive

The General Data Protection Regulation (GDPR) and the ePrivacy Directive (ePD) govern data collection and processing in the European Union (EU). The GDPR defines the data subject (user), controller (typically a website operator),

[6] WebAssembly is a binary instruction format designed as a fast, safe compilation target for languages like C/C++. It provides JavaScript interoperability and runs in web browsers [17].

[7] All code must be included within the extension package which is published in the extension stores.

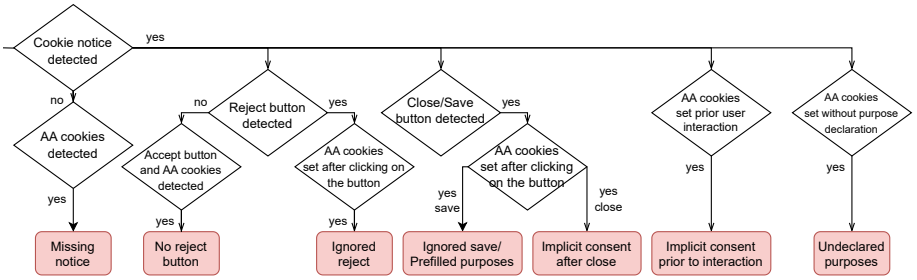


Figure 2.4: Decision tree for violations in cookie notices by Bouhoula et al. [4].

processor (e.g., a cloud provider), and third parties as key entities involved in data collection and processing. It regulates controllers within the EU as well as those who handle the personal data of EU residents. Both laws only regulate personal data, which is specifically defined in Article 4 of the GDPR as any information related to an "identified or identifiable natural person". Browser cookies that are not strictly necessary to provide the service, on the other hand, are covered by Article 5(3) of the ePD. The article requires user consent for storing and accessing data on the users' device, unless strictly necessary for providing a service [18].

For a controller to lawfully process personal data, at least one of the six possible legal bases defined in Article 6 of the GDPR must be satisfied. These legal bases include consent, contractual necessity, legal obligation, vital interests, public interest, and legitimate interests. However, in the particular context of websites using personal data and cookies for monetization, only the basis of user consent can be used [18]. Consent must be a "freely given, specific, informed, and unambiguous indication of the data subject's agreement to the processing of personal data relating to him or her" (Recital 32 of the GDPR). This implies that consent on websites has to be a voluntary choice by the subject, with equal ease for acceptance and rejection (*freely given*), should always be tailored to a single collection and processing purpose (*specific*), based on clearly communicated practices of data collection and processing (*informed*), and explicitly provided by the user, such as by clicking on a checkbox or signing up for a newsletter (*unambiguous*) [18]. Fig. 2.4 presents the possible privacy violations in cookie notices as a summary decision tree.

Non-compliance with the GDPR can result in severe penalties. Supervisory authorities can impose fines of up to €20 million or 4% of the organization's global annual turnover, whichever is higher [5].

2.4 Dark patterns

Cookie notices can include deceptive user interface (*UI*) designs that aim to manipulate user behavior and choices. Gray et al. [19] define dark patterns as "instances where designers use their knowledge of human behavior (e.g., psychology) and the desires of end users to implement deceptive functionality that is not in the user's best interest". The authors also identify the five primary types: *nagging* (repetitive interruptions that divert users from their intended tasks), *obstruction* (deliberate complication of user paths to discourage specific actions), *sneaking* (concealment or delayed revelation of relevant information to users), *interface interference* (UI manipulation that unfairly prioritizes certain options over others), and *forced action* (mandating specific user actions to access desired features or content) [19].

Nouwens et al. have found such dark patterns in cookie notices to significantly influence user behavior towards providing consent [6]. Furthermore, according to Gray et al. [20], some dark patterns could be considered forbidden under the GDPR in specific jurisdictions.

Chapter 3

Related work

Several studies have investigated website violations of privacy regulations. The impact of the 2018 GDPR has been a particular focus of research. The meta-study by Kretschmer et al. [1] provides an overview of the observed, quantitative effects on cookie compliance and privacy policies.

3.1 Large-scale analyses

We will first present some studies that developed large-scale approaches to investigate general trends in GDPR compliance. Many of them had a narrow scope either in the technical methods (e.g., restricted to specific CMPs, keyword-based analysis), or the investigated characteristics (e.g., only implicit consent, or existence of opt-out functionality). Finally, we will highlight two studies of special relevance to our work on CookieAudit.

Trevisan et al. [21] analyzed websites for implicit consent. In their study, they did not interact with cookie notices and considered third-party cookies to be AA cookies if they came from known advertising domains, which they classified using advertising lists by Ghostery and Disconnect. They found that 49% of websites created such AA cookies before any notice interaction. Nouwens et al. [6] tested for implicit consent, forced action, and pre-checked options on 680 websites popular in the UK and using specific CMPs. The researchers found only 11.8% of websites compliant. Kampanos et al. [22] analyzed the presence of cookie notices and opt-out functionality (i.e., a reject button) on 17k websites from the UK and Greece. They used a crawler with automated notice detection based on EasyList selectors and keyword-based consent option detection.

Bollinger et al. [3] investigated cookie consent practices and GDPR compliance on websites using specific Consent Management Platforms (CMPs). They observed that 69.7% of websites set non-essential cookies before ob-

taining user consent. For their analysis, they developed a gradient boosting model to classify cookies into Essential, Functional, Analytics, and Advertising categories. Building on this, Bouhoula et al. [4] created a fully automated, CMP-independent approach for large-scale analysis of GDPR cookie compliance. In addition to building on the cookie classification model by Bollinger et al., they developed NLP models to analyze the text of cookie notices and their interactive elements. In their comprehensive set of 97k popular websites targeting EU users, they found that 65.4% of websites collected user data after an explicit rejection.

3.2 Inspection of individual web pages

There is a need by website operators but also regulators for auditing tools that verify compliance of websites. Next, we will give an overview of the existing tools while also highlighting their particular limitations.

3.2.1 CookieAudit

Kubicek and Zanga et al. developed the initial version of the browser extension CookieAudit [2]. It aims to find violations of privacy regulations of websites in cookies and cookie notices. Building on the work by Bollinger et al. [3], the extension detects non-essential cookies being set without proper consent. Furthermore, it identifies missing reject buttons in cookie notices and pre-selected consent options by using keyword-lists. However, this version of CookieAudit was limited in several ways. It required the user to interact with the cookie notice and website during the entire scan, and it relied on hard-coded keyword-lists which are restricted to English and German. Moreover, some analysis functionality of cookie notice settings was restricted to specific CMPs. The old CookieAudit version thus lacks in universal applicability and usage ergonomics.

3.2.2 European Data Protection Board Website Auditing Tool

EDBP WAT is an auditing tool developed for the European Data Protection Board [8]. It uses a custom Chromium interface to analyze how sites collect and store browser data, as well as inspect network traffic. Users can record different scenarios, e.g., the rejection of cookies and subsequent manual interaction with the website. EDBP then categorizes the collected cookies, local storage, web beacons[1] and HTTP requests by matching against tracker databases. The tool generates customizable reports to export the findings.

[1] A web beacon (or tracking pixel) is a method for tracking users. Visitors can be monitored by including an image (often invisible) from an external server inside the website [23].

3.2.3 Privacy Pioneer

Zimmeck et al. [24] designed *Privacy Pioneer*, a browser extension for real-time detection of GDPR-relevant data collection. The extension monitors web traffic by analyzing HTTP messages using the `webRequest` API. It searches the traffic for location information, monetization practices, tracking methods, and personal data. For this, it uses a combination of regular expression matching, lists of known tracker URLs, and a machine learning model trained to detect location data. Privacy Pioneer is limited to the Firefox browser and can be quite performance-intensive, as processing all traffic with ML models leads to slower browsing speeds. In contrast to CookieAudit, Privacy Pioneer does not analyze cookies or cookie notices.

3.2.4 Cookie Glasses

Matte et al. [7] have investigated the compliance of cookie banners implementing IAB Europe's Transparency and Consent Framework (TCF) and found that they did not always respect the users' choice. As part of the research, the browser extension *Cookie Glasses* was developed. It aimed to help website developers verify their compliance. It was restricted to TCF cookie notices and in particular TCF version 1 (v2.2 is the most recent one). As of July 2024, the Cookie Glasses extension no longer works and the development seems to be abandoned.

Chapter 4

Application flow

CookieAudit aims to analyze websites for the violations and dark patterns noted in Sections 2.3 and 2.4. This chapter outlines the user's view of the CookieAudit extension, from the start of a scan to the report generation. The following sections describe each phase of the extension's analysis, including the necessary user interactions. This overview provides context for the implementation details discussed in Chapter 5.

4.1 Installation and onboarding

CookieAudit can be installed in modern Chromium-based web browsers such as Chrome and Edge.[1] Furthermore, because of the small differences between Chromium-based browsers and Firefox in the WebExtensions APIs used by CookieAudit, we expect that only minimal changes will be necessary to port the extension to Firefox. Upon installation, CookieAudit automatically opens an onboarding document (Fig. 4.1) explaining the usage and functionality of the extension. This document can also be accessed later via the "Help Page" link in the extension popup shown in Fig. 2.3.

4.2 Scan start and notice selection

After opening the targeted website, the user can start a new CookieAudit scan with the "Start Scan" button in the initial extension popup shown in Fig. 2.3. Next, the CookieAudit notification presented in Fig. 4.2 asks the user to select the cookie notice. Using an element picker, the notice element is selected by hovering over it and clicking. After the selection is confirmed in the custom context menu on the top left of the webpage, CookieAudit

[1] At the time of writing, CookieAudit has not yet been published to the Chrome Web Store. Currently, the extension has to be installed by manually loading the extension bundle into the browser.

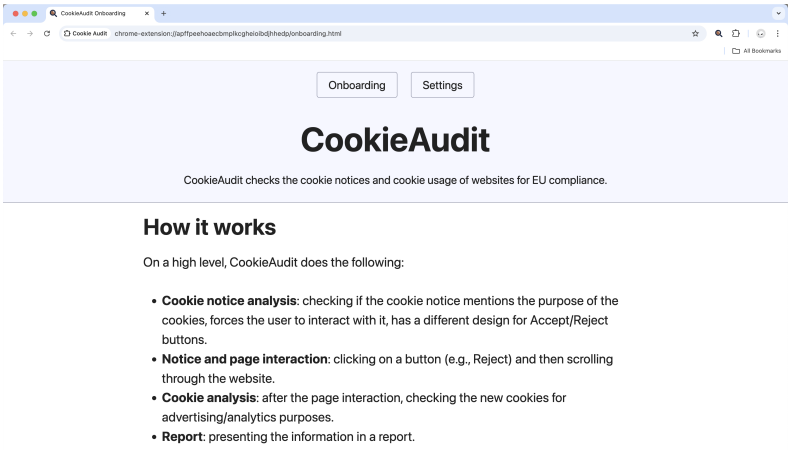


Figure 4.1: The extension onboarding page contains a guide on the usage of CookieAudit.

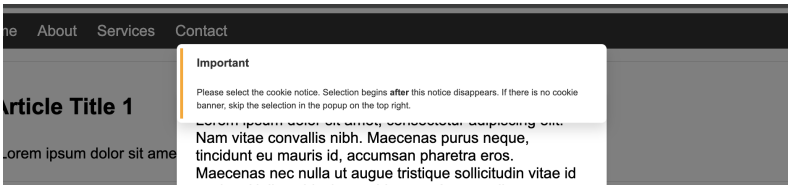


Figure 4.2: CookieAudit notifications inform the user about the progress of the scan.

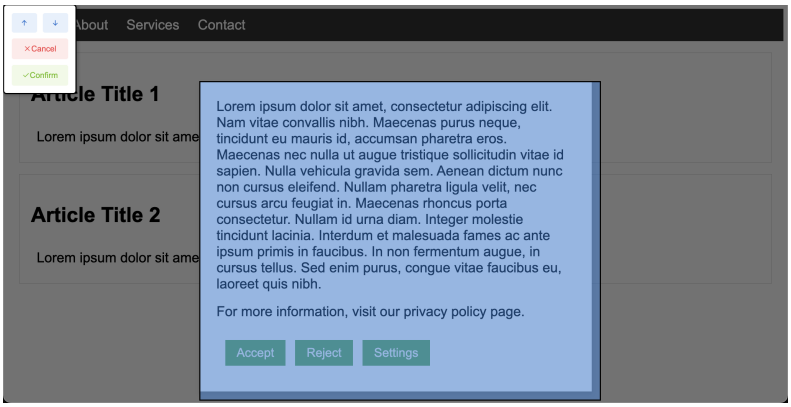


Figure 4.3: Cookie notice selected by the user, awaiting confirmation.

extracts the text and links from the cookie notice and stores them. Figure 4.3 shows a selected cookie notice before confirmation via the context menu.

4.3 Model download and notice analysis

The extension uses two *Bidirectional Encoder Representations from Transformers* [25] (*BERT*) based models by Bouhoula et al. [4] for the analysis of the

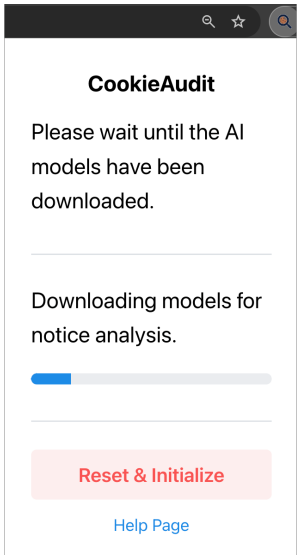


Figure 4.4: The CookieAudit extension popup shows the download progress of the remotely hosted BERT models.

cookie notice content. If CookieAudit is run for the first time, it will need to download the remotely hosted models and store them in the browser cache. The download progress is indicated in the extension popup shown in Fig. 4.4. Once available, CookieAudit uses the *purpose detection model* to verify whether the cookie notice properly specifies the purposes of the data collection and processing, as required by the GDPR. The second model is used to classify the links and buttons of the cookie notice into their uses: accept, reject, close, save, settings, and other. If a settings button is found, CookieAudit clicks it and may prompt the user to manually select the cookie settings element.[2] The process of selecting the cookie settings is identical to the cookie notice selection explained in Section 4.2. The settings notice's contents are analyzed in the same way as the cookie notice's contents.

4.4 Automatic interaction and cookie analysis

At this point, CookieAudit starts verifying the cookie compliance of the website. That is, the extension will start monitoring and classifying all cookies being set by the website into their respective functionalities, which are necessary, functional, analytics, and advertising. For the cookie classification, CookieAudit uses the ensemble model by Bollinger et al. [3]. Additionally, CookieAudit will interact with the cookie notice and website to imitate user behavior. For example, if a cookie notice contains a reject button, CookieAudit

[2]The criteria for automatic identification or manual user selection of the cookie settings are detailed in Section 5.3.3.

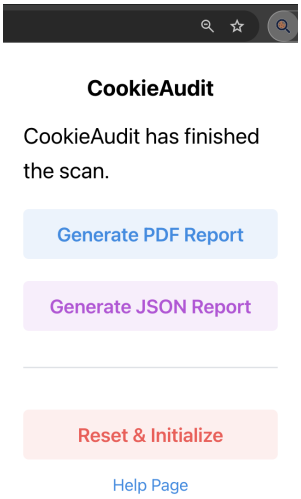


Figure 4.5: The reports can be generated via the CookieAudit extension popup.

will click it, visit three subpages of the website, and scroll through them. Any cookies that are classified as analytics or advertising during this process are stored as potential compliance violations. According to Urban et al. [26], subpages of websites set 36% more cookies than their respective landing pages. Therefore, it is crucial to visit subpages to get a more representative image of a website's cookie usage.

CookieAudit automatically inspects cookie notices for dark patterns. This includes comparing the dominant colors of interactive elements and verifying whether the notice blocks interaction with the rest of the website. Dark patterns were defined in Section 2.4.

4.5 Report generation

Upon completion of the scan, a PDF report and a machine-readable report based on the JavaScript Object Notation (*JSON*) can be generated. The Fig. 4.5 shows the updated extension popup, where both reports can be created.[3]

[3]The reports are generated in the user's browser. However, after clicking the generation button, the corresponding report appears to be downloaded and is added to the user's downloads folder.

Chapter 5

Implementation

5.1 User interface

In CookieAudit, users can initiate a website scan, stop it, or download a report summarizing the scan results via the extension popup. Additionally, CookieAudit displays real-time information about the scan's progress. For developing the popup's user interface, we chose to use the React JavaScript library. React offers a declarative programming model that keeps the displayed data synchronized with the underlying data storage.

5.2 Cookie notice selection

CookieAudit is designed to automatically analyze and interact with arbitrary cookie notices. The initial identification of which HTML element represents the cookie notice could be done either automatically or manually by the user. Automatic notice selection, e.g., based on heuristics or NLP models, can be error-prone. On the other hand, a manual selection could provide better accuracy with minimal user effort. Next, we will give an overview of both approaches and their limitations.

5.2.1 Automatic selection

Any approach to automated notice selection will always be limited in its accuracy. For example, Bouhoula et al. [4] achieve a precision of 100% and a recall of 86.9%, even though the authors improve in both metrics by building upon the approaches for automatic selection of previous studies. Those issues fundamentally stem from the many ways of implementing cookie notices in HTML and JavaScript. Nevertheless, we will present some of those heuristics.

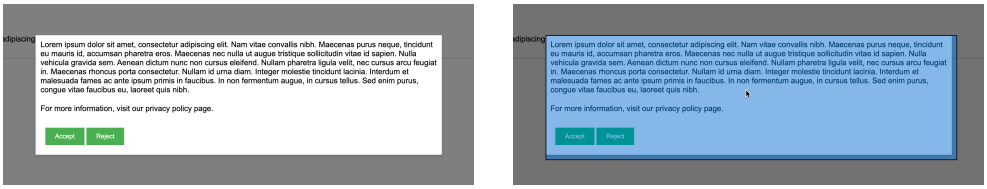


Figure 5.1: Manual notice selection.

Crowdsourced selectors

HTML elements can be identified with *selectors*. Crowdsourced efforts, such as EasyList Cookie [27], maintain lists of cookie notice selectors. Automated crawlers by Kampanos et al. [22] and Bouhoula et al. [4] have used EasyList to find potential cookie notices on websites.

Stack level

The *z-index* is a CSS property of HTML elements that makes elements with a high index appear above elements with a low index. Because a user should quickly see a cookie notice, websites usually use a high *z-index* on the notice to let it cover the rest of the website. This heuristic has been used by Khandelwal et al. [28] and Bouhoula et al. [4].

5.2.2 User-aided selection

CookieAudit is intended for semi-automated website analysis. We can therefore rely on user help by using a cookie notice picker tool. To specify the element containing a notice, users only needs to hover the mouse over it and then confirm the selection. An example can be seen in Fig. 5.1.

The picker is implemented as a *content script*. As explained in Section 2.1.2, content scripts are JavaScript programs that can interact with the web page.

The activated picker tool listens for cursor movement by the user. On movement, it reads the coordinates of the cursor and determines the HTML element at that position. The picker changes the styling of the determined element to provide visual feedback. CookieAudit needs users to select the element corresponding to the complete cookie notice. Occasionally, the desired element is only a container whose area is filled out by the children (see Fig. 5.2). In this case, every click inside the notice will select a child element rather than the entire notice container. Therefore, users won't be able to select the whole notice by simply hovering the cursor. As a solution, we implement a context menu that appears after the user has clicked on an element. With the menu, the user can traverse up the DOM tree to the element containing the whole cookie notice. In the end, the selection can be confirmed via the context menu.

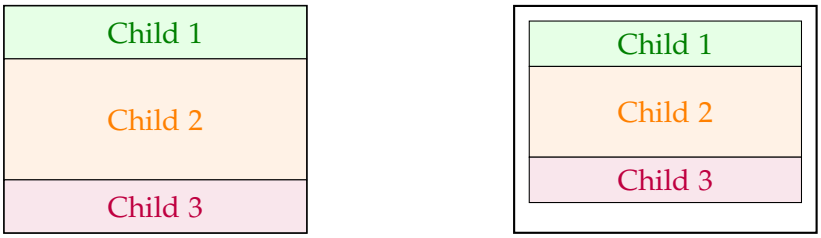


Figure 5.2: Any location in the left notice will map to one of the children. Only the white area in the right notice will be mapped to the whole notice.

The complexity and variety of both HTML and JavaScript implementation details results in many possible ways of implementing cookie notices. Because of this, we had to handle several edge cases which we encountered during the manual testing of CookieAudit. Next, we will present our solutions.

Inline frame

An inline frame (`iframe`) allows developers to embed another HTML context by specifying an external web address [29]. Some websites implement cookie notices with an `iframe` that refers to a Consent Management Platform. To enable users to select the cookie notice, we inject the *picker tool* content script not only into the original web page, but also into all `iframe` elements.

Shadow DOM

A shadow DOM separates the elements inside of it from the influence of the main document's JavaScript and CSS [30]. Some developers use this feature to integrate cookie notices into their web page. This encapsulation prevents our picker content script from directly accessing the cookie notice elements. To address this, we implement a solution during user notice selection. As described before, we continuously query for the element that is located at the user's cursor coordinates. When a user hovers over an element within a shadow DOM, this query will return the root of the shadow DOM (*shadow root*). We then check if the element is indeed a shadow root by reading the `shadowRoot` property of the HTML element. If so, we explicitly enter the shadow DOM context and query for the element at the user's cursor coordinates once again.

Dialog element

The `dialog` element is an HTML element that can be used to build dialog boxes such as cookie notices. When opened, it is displayed above the other elements of the website and covers elements with arbitrary `z-index` values. Multiple `dialog` elements are stacked based on their opening order, with the most recent dialog displayed on top. If used to display a cookie notice, a

dialog element would also hide our picker tool context menu. Therefore, we have implemented the menu with a `dialog` element. By opening our dialog after the cookie notice renders, we ensure that the menu will be visible and accessible to CookieAudit users.

5.3 Cookie notice analysis

For the analysis of a selected cookie notice, we rely on the extraction and classification functionality developed by Bouhoula et al. [4]. We use their NLP models to detect declared cookie purposes in the cookie notice text and to classify interactive elements of the notice for later interaction. In case of a detected *Settings* button, we additionally analyze the corresponding settings page.

5.3.1 Extraction

To extract the cookie notice text, we iterate through all visible elements inside the notice and concatenate their text content. For nested elements, which are common in HTML's tree-like structure, we recursively repeat this process.

To enable automated interaction with cookie notices, CookieAudit needs to identify all clickable elements. We use the criteria proposed by Bouhoula et al. [4]. First, we consider only elements of type `div`, `span`, `a`, `button` or `input` that fulfill one of the following criteria: has the accessibility (ARIA) role `button`, has the attribute `onclick` or has a non-negative `tabindex`. Furthermore, we consider elements that change the mouse to a pointer when hovered over. The last criterion was added based on manual testing, as it captured some interactive elements missed by the other rules in rare cases.

5.3.2 Classification

According to the GDPR, users need to be able to provide informed and explicit consent. Therefore, a cookie notice needs to (i) declare the purposes of its data collection and processing and (ii) provide interactive elements for the user to express his explicit consent (*Accept* button) or rejection (*Reject* button). Bouhoula et al. [4] trained the two BERT models [25], the *purpose detection* model and the *interactive elements* model, for both classification problems. Next, we will explain how we integrate and use the models.

Purpose detection model

The purpose detection model classifies sentences into either Purposes for Analytics/Advertising (profiling, advertising, custom content, analytics, and social media features) or Other Purposes (essential functionalities, offering service, and website/UX enhancement). Because the models only work on

English input, CookieAudit first translates (if necessary) the cookie notice text using a public translation API. The translated text is then classified with the browser library Transformer.js [31] which makes it possible to efficiently[1] run BERT models on the client.

Interactive elements model

CookieAudit has to understand the functionality of the interactive cookie notice elements to be able to automatically interact with them. The interactive elements model by Bouhoula et al. classifies the texts of buttons as either accept, reject, close, save, settings and other. Because it is also a BERT model we can run it, after a translation of the texts, equivalently to the purpose detection model.

Cookie classification model

We use the CookieBlock model by Bollinger et al. [3] which classifies cookies into the four purposes Necessary, Functional, Analytics, and Advertising. The model relies on features such as one-hot encoding of common cookie names and domains, the presence of specific data types, the cookie's expiry date and entropy, or the edit distance between cookie updates. The ensemble model was trained on a dataset of cookies that were collected from crawling 27k websites which employed one of the CMPs: OneTrust, Cookiebot, or Termly. These CMPs provide labels for individual cookies.

5.3.3 Exploration

Cookie notices can offer additional settings. The settings may contain a declaration of cookie purposes, or interactive elements, e.g, to reject specific cookie categories or even all non-necessary cookies. Therefore, CookieAudit needs to open and analyze the settings.

If a *Settings* button is available in a cookie notice, clicking it may change the content of the current notice to display the settings, open a new settings notice or open a separate web page with the settings. Similar to the approach implemented by Bouhoula et al. [4], CookieAudit clicks on all interactive elements that were classified as *Settings*. If no interactive elements were classified as *Settings*, CookieAudit alternatively uses three *Other* elements. In particular, the *Other* elements are sorted by their y-coordinates, such that elements that are located on the bottom of the cookie notice are used. This

[1] We have found the average time of a sentence classification to be 209ms. To determine this, we benchmarked the classification of 1187 sentences using the purpose detection model in a Chrome extension using Transformers.js. The sentences are from a dataset by Santos et al. [32] and were also used to train the purpose detection model. The benchmark was done on a MacBook Pro with an Intel i5 7267U 3.10GHz CPU and 16 GB memory while no other user space programs were running.

simple heuristic is based on the observation that the *Settings* button is often located at the bottom. After the *Settings* have been opened and identified, CookieAudit will again proceed as for the original cookie notice, retrieving and analyzing the text and interactive elements. CookieAudit saves the elements of type *Reject*, *Close* and *Save Settings* to the extension storage. The stored entries are pairs of selectors containing the *Settings* element selector, and the second layer element selector.

Based on the work by Bouhoula et al. [4], we implemented the detection and handling of the following three scenarios:

New cookie notice

The cookie notice settings can be displayed inside a new notice that appears, e.g., above the original cookie notice. CookieAudit detects this situation if one of the following criteria holds:

- a query for the cookie notice with the selector returns `null`,
- the cookie notice is not visible anymore (e.g., if the notice has or inherits an `opacity` of 0, has or inherits the CSS `visibility` of hidden or collapse, or has an area of 1 pixel or less),
- the notice still exists and is visible, but *neither* its text content nor the dimensions (height and width) have changed after the interactive element click, or
- the cookie notice is covered by another element. CookieAudit runs `elementFromPoint()` at the coordinates of the cookie notice center. If the returned element is not the notice or contained in the notice, we consider the notice covered.

To identify the settings element and start the analysis, CookieAudit activates the *Cookie Notice Picker* (as described in Section 5.2.2) and lets the user manually select the settings notice.

New web page

CookieAudit stores the URL of the web page before the interactive element click and compares it to the URL of the web page after the click. If it has changed, CookieAudit classifies the outcome as a new notice on a separate web page. In this case, CookieAudit starts the cookie notice picker. The picker can be used to select both settings that are part of the normal flow of the web page, or displayed as a hovering dialog. After the user selection and subsequent settings analysis, CookieAudit returns to the URL where the scan was started.

Same cookie notice

If neither the criteria of a *New Cookie Notice* nor *New Web Page* apply, CookieAudit assumes the settings to be now contained in the original cookie notice. This effectively means that the original cookie notice is still visible, the contents, or dimensions have changed and the URL of the web page is still the same. CookieAudit directly starts the settings notice analysis without any user interaction.

5.4 Cookie analysis

As described in Section 4.4, CookieAudit stores and classifies the cookies set by a website during a scan. This data is then used to assess the compliance of the cookie notice and the website.

5.4.1 Cookie monitoring

CookieAudit uses the `cookies.onChanged` event object from the WebExtensions API to monitor cookies set or updated by the website. When a cookie is set for the first time, the extension stores all relevant data and classifies the cookie. If a cookie is updated, CookieAudit updates the stored data and reclassifies the cookie based on the most recent information.

5.4.2 Cookie classification

We employ the CookieBlock model by Bollinger et al. [3], which classifies cookies into four categories: Necessary, Functional, Analytics, and Advertising. The model uses features such as one-hot encoding of common cookie names and domains, the presence of specific data types, the cookie's expiration date and entropy, and the edit distance between cookie updates. The ensemble model was trained on a dataset of cookies collected from 27k websites that were using one of the CMPs: OneTrust, Cookiebot, or Termly.

5.5 Dark pattern detection

Deceptive user interface designs that aim to manipulate user behavior and choices can be used in cookie notices. Nouwens et al. have found such dark patterns in cookie notices to significantly influence user behavior towards providing consent [6]. Because some dark patterns could be considered forbidden under the GDPR in specific jurisdictions [20], we include detection of dark patterns as part of the CookieAudit extension. Specifically, we consider the same three characteristics as Bouhoula et al. [4] in our analysis: "No reject button", "Interface interference", and "Forced action". Next, we

will describe how CookieAudit detects "Interface interference" and "Forced action".

5.5.1 Interface interference

Stöver et al. [33] define interface interference as "design (color, font, size) of the buttons for accepting and rejecting cookies are not equivalent." To find cases of interface interference, CookieAudit compares design characteristics of *positive consent options* (accept button) with all *negative consent options* (reject, save, and close buttons).

Typography differences of buttons can be detected by comparing the CSS attributes of the HTML elements, e.g., `font-size`. The dominant color of a button is influenced by many potential factors, e.g., transparency of a button, background images, or gradients. Therefore, CookieAudit captures a screenshot of the visible web page using the WebExtensions API and analyzes it to reliably determine the dominant colors of the buttons After the screenshot is taken, the extension crops the image for all buttons according to their coordinates and determines the dominant color for each consent option with a k-means analysis. Once all dominant colors have been identified, CookieAudit calculates the color differences between the positive and negative consent options. The Euclidean distance over RGB (red, green, blue) color tuples would not accurately represent the perception of humans. Instead, the extension uses the ΔE_{ITP} color difference metric [34] which is based on a uniform color space, that is, "a color space in which equivalent numerical differences represent equivalent visual differences, regardless of location within the color space" [35]. The ΔE_{ITP} metric is defined such that a difference below 1 is not noticeable for the human eye. We consider two colors to be mismatched when their ΔE_{ITP} difference is higher than 25.

5.5.2 Forced action

Stöver et al. [33] define "Forced action" as the requirement for a user to first engage with a cookie notice before entering the website. In its most extreme form, this would be a "cookie wall" which forces the user to first accept all cookies and tracking technologies [4]. Gray et al. [20] considers "Forced action" to be a dark pattern. Bouhoula et al. [4] argue that such a design is not necessarily intentional deception, as it may even raise the privacy awareness of users, according to Bakos et al. [36]. CookieAudit will report any detected "Forced Action" as a dark pattern.

"Forced Action" prevents the user to interact with the rest of the website, e.g., clicking on links outside the cookie notice. First, CookieAudit samples 10 links from the website, which are located in the viewport and are visible according to their CSS attributes. Because the cookie notice body may

cover some links but still allow most website interaction,[2] CookieAudit selects, if possible, links that are not hidden by the cookie notice body. Next, CookieAudit will determine whether the links are unresponsive while the cookie notice is open, but clickable after CookieAudit has provided positive consent, i.e., clicked the Accept button. For a browser extension, there is no general method to determine if a link is clickable by the user. Instead, CookieAudit will approximate the "clickability" by determining whether a link is covered by another element, e.g., by a semi-transparent overlay that blocks user clicks. CookieAudit reports forced action if at least one link which is covered before any consent is given,[3] is accessible after the cookie notice is closed by CookieAudit.

5.6 Managing page loads and changes

In the course of a website scan, CookieAudit repeatedly reloads the website and visits multiple subpages. The extension always needs to wait until all relevant content has been loaded, e.g., CookieAudit can only start analyzing the text and links in the cookie notice once all of them have been rendered in the browser. CookieAudit uses the WebExtensions API `tabs.onUpdated` to determine when a website is fully loaded and rendered. The `tabs.onUpdated` event object informs a browser extension about the loading state of the initial HTML document and its immediate resources (CSS, JavaScript, and hard-coded images). However, the event object does not inform CookieAudit when websites dynamically add content, which is also possible after the initial HTML document is loaded. The loading state, that CookieAudit receives from the WebExtensions API, is also indicated in the Chrome browser interface with a loading icon. Fig. 5.3 shows a website dynamically adding a CMP-based cookie notice after Chrome considers the website to be loaded. Therefore, CookieAudit must detect dynamic changes in a website's content after the initial HTML document has finished loading.

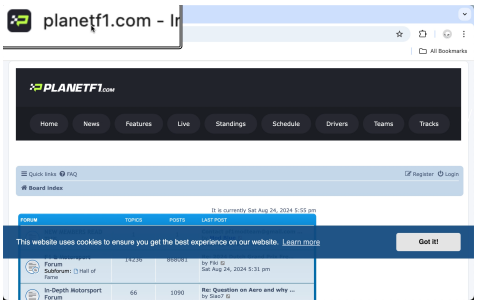
CookieAudit needs to watch the whole DOM subtree rooted in the document body for changes because any update could be relevant for the website analysis. The `MutationObserver` interface, which is implemented in all major browsers, can be used to efficiently watch for mutations in a website's DOM tree [37]. We configure the `MutationObserver` to monitor the document body subtree for additions or removals of child elements. We consider the website to be loaded when no new changes are observed for two seconds. Because some websites continuously make dynamic DOM changes, e.g., loading and adding new posts to a timeline, we add a general timeout of ten seconds. During manual testing, we observed the timeout of ten seconds to be effective.

[2] For example, a cookie notice can cover the navigation header while the body of the website is still accessible.

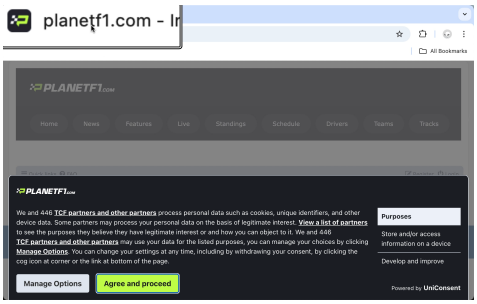
[3] All default counts and thresholds can be configured by the user.

In all cases where it was reached because of continuous DOM changes, the websites had already finished loading all required elements[4] for CookieAudit.

(a) The spinning loader indicates that the website is loading the initial HTML document.



(b) The initial HTML document has been loaded.



(c) Dynamic changes are not indicated by a change in the loading state.

Figure 5.3: Website loading sequence: initial page load, HTML rendering completion, and optional dynamic content update.

5.7 Report creation

After the scan, CookieAudit compiles two reports summarizing the collected data and analysis. The first report is a PDF file created using the JavaScript-based pdfmake library. Pdfmake is a document generation library for the browser that enables the convenient styling and layout of document contents, table layouts with header repetition across page breaks, and automatic download functionality that doesn't require the user to manually "print" a report document.

To generate the machine-readable JSON report, CookieAudit collects all relevant data in a JavaScript object, converts it into a string, and writes it into

[4]Depending on the stage of the scan, not only the cookie notice is required. For example, the navigation header is needed to find subpages of the website during the consent options testing.

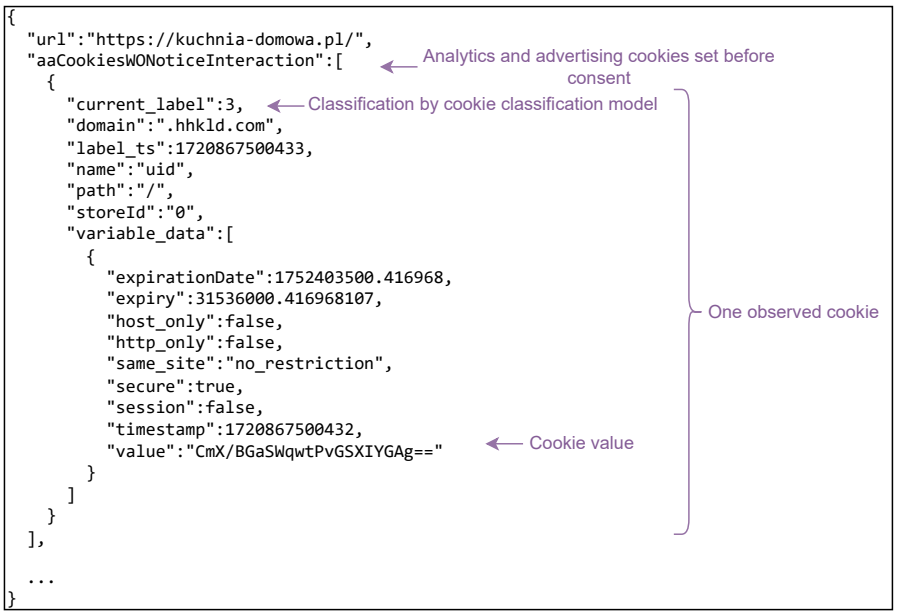


Figure 5.4: A partial extract of the JSON report. It includes a potential advertising cookie.

a `.json` text file. The conversion is straightforward because the JSON data format structure is derived from JavaScript objects. The JSON report contains more detailed data collected by CookieAudit. For example, Fig. 5.4 shows an extract of a JSON report from manual testing (see Chapter 6) which contains a potential advertising cookie that was set prior to consent.

Chapter 6

Testing of CookieAudit

To evaluate CookieAudit, we created two distinct sets of websites based on the collection crawled by Bouhoula et al. [4]. We randomly sampled web pages where their crawler detected a cookie notice and successfully conducted an analysis.

The first set comprises 50 websites from 5 countries (France, Germany, Poland, Ireland, and the USA), representing different popularity levels according to the Chrome User Experience (*CrUX*) report [38]. Specifically, we included web pages from CrUX ranks 1k, 5k, 10k, 100k, and 500k.[1] The second set consists of 20 websites, each using one of the 20 most popular Consent Management Platforms.

We manually tested the CookieAudit browser extension on these websites from a location in Germany and compared the results stored in the machine-readable JSON reports with the corresponding findings of Bouhoula et al. [4]. Due to the limited dataset size, many types of privacy violations and dark patterns are represented in single digits and cannot be meaningfully compared. We computed the precision and recall of CookieAudit, assuming the crawling results of Bouhoula et al. [4] as the ground truth. The results are presented in Table 6.1.

Website changes between the original crawling date and the time of manual analysis likely explain some differences in button detection and implicit consent identification. Additionally, CookieAudit's approximate method for assessing link clickability (see Section 5.5.2) likely contributes to poorer forced action detection compared to the *OpenWPM* framework [39] used by Bouhoula et al. [4], which can directly verify link interactivity.

[1] For example, the CrUX rank 1k contains the 1000 most popular web pages of a given country.

Criterion	Precision	Recall
Functionality		
Reject button detection	0.82	1.0
Violations and Dark Patterns		
Implicit consent prior to interaction	1.0	0.78
Forced action	0.86	0.68
Interface interference	1.0	1.0

Table 6.1: Precision and recall of CookieAudit detection in comparison to the crawler by Bouhoula et al. [4].

Bibliography

[1] M. Kretschmer, J. Pennekamp, and K. Wehrle, "Cookie banners and privacy policies: Measuring the impact of the gdpr on the web," *ACM Transactions on the Web*, vol. 15, pp. 1–42, Jul. 2021, ISSN: 1559-1131, 1559-114X. DOI: 10.1145/3466722. [Online]. Available: https://dx.doi.org/10.1145/3466722.

[2] K. Kubicek, D. Bollinger, A. Zanga, C. Cotrini, and D. Basin, "CookieBlock & CookieAudit: Fixing cookie consent with ML (poster)," Boston, MA: USENIX Association, Aug. 2022.

[3] D. Bollinger, K. Kubicek, C. Cotrini, and D. Basin, "Automating cookie consent and GDPR violation detection," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 2893–2910, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/bollinger.

[4] A. Bouhoula, K. Kubicek, A. Zac, C. Cotrini, and D. Basin, "Automated large-scale analysis of cookie notice compliance," in *USENIX Security Symposium*, 2023.

[5] I. Sanchez-Rola *et al.*, "Can i opt out yet?" *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, vol. Not available, Not available, Jul. 2019, ISSN: Not available. DOI: 10.1145/3321705.3329806. [Online]. Available: https://dx.doi.org/10.1145/3321705.3329806.

[6] M. Nouwens, I. Liccardi, M. Veale, D. Karger, and L. Kagal, "Dark patterns after the GDPR: Scraping consent pop-ups and demonstrating their influence," *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, Apr. 2020. DOI: 10.1145/3313831.3376321. [Online]. Available: https://dx.doi.org/10.1145/3313831.3376321.

[7] 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*, 2020. arXiv: 1911.09964 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1911.09964.

[8] J. Gorin. "EDPB website auditing tool." (Jan. 2024), [Online]. Available: https://code.europa.eu/edpb/website-auditing-tool (visited on 08/02/2024).

[9] Cookie Information A/S. "Cookie compliance audit tool." (Nov. 2019), [Online]. Available: https://chromewebstore.google.com/detail/mginbgceeeielfenknclfcfegblhpkpf (visited on 08/03/2024).

[10] Chrome for Developers. "Manifest V3." (), [Online]. Available: https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3 (visited on 07/30/2024).

[11] M. Frisbie, "Browser extension architecture," in *Building Browser Extensions: Create Modern Extensions for Chrome, Safari, Firefox, and Edge*. Berkeley, CA: Apress, 2023, ISBN: 978-1-4842-8725-5. DOI: 10.1007/978-1-4842-8725-5_4. [Online]. Available: https://doi.org/10.1007/978-1-4842-8725-5_4.

[12] K. Purdy. "Chrome's Manifest V3 and its changes for ad blocking, are coming real soon." (Aug. 2024), [Online]. Available: https://arstechnica.com/gadgets/2024/08/chromes-manifest-v3-and-its-changes-for-ad-blocking-are-coming-real-soon/ (visited on 08/06/2024).

[13] G. Huczynski, R. Hill, K. Resa, *et al.* "Chrome extension manifest v3 proposal." (Dec. 2018), [Online]. Available: https://github.com/uBlockOrigin/uBlock-issues/issues/338 (visited on 08/06/2024).

[14] R. Hill. "About Google Chrome's "This extension may soon no longer be supported"." (Aug. 2024), [Online]. Available: https://github.com/gorhill/uBlock/wiki/About-Google-Chrome's-%22This-extension-may-soon-no-longer-be-supported%22 (visited on 08/06/2024).

[15] C. Gonzalez. "Bitwarden transitions from manifest v2 to v3." (May 2024), [Online]. Available: https://bitwarden.com/blog/bitwarden-manifest-v3/ (visited on 07/30/2024).

[16] N. Carlini, A. P. Felt, and D. Wagner, "An Evaluation of the Google Chrome Extension Security Architecture," in *21st USENIX Security Symposium (USENIX Security 12)*, Bellevue, WA: USENIX Association, Aug. 2012, pp. 97–111. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/carlini.

[17] A. Haas *et al.*, "Bringing the web up to speed with WebAssembly," *ACM SIGPLAN Notices*, vol. 52, pp. 185–200, Jun. 2017, ISSN: 0362-1340, 1558-1160. DOI: 10.1145/3140587.3062363. [Online]. Available: https://dx.doi.org/10.1145/3140587.3062363.

[18] K. Kubicek, "Automated analysis and enforcement of consent compliance," Ph.D. dissertation, ETH Zurich, 2024. DOI: 10.3929/ethz-b-000662039.

[19] C. M. Gray, Y. Kou, B. Battles, J. Hoggatt, and A. L. Toombs, "The dark (patterns) side of ux design," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18, Montreal QC, Canada: Association for Computing Machinery, 2018, pp. 1–14, ISBN: 9781450356206. DOI: 10.1145/3173574.3174108. [Online]. Available: https://doi.org/10.1145/3173574.3174108.

[20] C. M. Gray, C. Santos, N. Bielova, M. Toth, and D. Clifford, "Dark Patterns and the Legal Requirements of Consent Banners: An Interaction Criticism Perspective," *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, May 2021. DOI: 10.1145/3411764.3445779. [Online]. Available: https://dx.doi.org/10.1145/3411764.3445779.

[21] M. Trevisan, S. Traverso, E. Bassi, and M. Mellia, "4 years of eu cookie law: Results and lessons learned," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, pp. 126–145, Apr. 2019. DOI: 10.2478/popets-2019-0023. [Online]. Available: https://dx.doi.org/10.2478/popets-2019-0023.

[22] G. Kampanos and S. F. Shahandashti, "Accept all: The landscape of cookie banners in Greece and the UK," in *IFIP International Conference on ICT Systems Security and Privacy Protection*, Springer, 2021, pp. 213–227.

[23] R. M. Smith. "The web bug FAQ." (Nov. 1999), [Online]. Available: https://w2.eff.org/Privacy/Marketing/web_bug.html (visited on 08/02/2024).

[24] S. Zimmeck *et al.*, "Website Data Transparency in the Browser," *Proceedings on Privacy Enhancing Technologies*, vol. 2024, pp. 211–234, Apr. 2024, ISSN: 2299-0984. DOI: 10.56553/popets-2024-0048. [Online]. Available: https://dx.doi.org/10.56553/popets-2024-0048.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[26] T. Urban, M. Degeling, T. Holz, and N. Pohlmann, "Beyond the front page:measuring third party dynamics in the field," in *Proceedings of The Web Conference 2020*, ser. WWW '20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 1275–1286, ISBN: 9781450370233. DOI: 10.1145/3366423.3380203. [Online]. Available: https://doi.org/10.1145/3366423.3380203.

[27] Fanboy, MonztA, Khrin, Yuki2718, and PiQuark6046. "EasyList." (2024), [Online]. Available: https://easylist.to (visited on 08/20/2024).

[28] R. Khandelwal, A. Nayak, H. Harkous, and K. Fawaz, "Automated cookie notice analysis and enforcement," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1109–1126.

[29] "<Iframe>: The inline frame element." (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe (visited on 07/09/2024).

[30] "Using shadow DOM." (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM (visited on 07/10/2024).

[31] Huggingface and J. Lochner. "Transformers.js." (2023), [Online]. Available: https://huggingface.co/docs/transformers.js/en/index (visited on 08/06/2024).

[32] C. Santos, A. Rossi, L. S. Chamorro, K. Bongard-Blanchy, and R. Abu-Salma, "Cookie banners, What's the Purpose?" *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, Nov. 2021. DOI: 10.1145/3463676.3485611. [Online]. Available: https://dx.doi.org/10.1145/3463676.3485611.

[33] A. Stöver *et al.*, "Website operators are not the enemy either-Analyzing options for creating cookie consent notices without dark patterns," 2022.

[34] "Objective metric for the assessment of the potential visibility of colour differences in television," Internation Telecommunication Union, Geneva, CH, Tech. Rep., Jan. 2019.

[35] X-Rite, Incorporated. "Color Glossary." (2024), [Online]. Available: https://www.xrite.com/learning-color-education/other-resources/glossary (visited on 08/22/2024).

[36] Y. Bakos, F. Marotta-Wurgler, and D. R. Trossen, "Does Anyone Read the Fine Print? Consumer Attention to Standard-Form Contracts," *The Journal of Legal Studies*, vol. 43, pp. 1–35, Jan. 2014, ISSN: 0047-2530, 1537-5366. DOI: 10.1086/674424. [Online]. Available: https://dx.doi.org/10.1086/674424.

[37] Web Hypertext Application Technology Working Group. "Interface MutationObserver." (Aug. 2024), [Online]. Available: https://dom.spec.whatwg.org/#interface-mutationobserver (visited on 08/25/2024).

[38] Chrome for Developers. "Overview of CrUX." (Feb. 2024), [Online]. Available: https://developer.chrome.com/docs/crux (visited on 08/06/2024).

[39] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of ACM CCS 2016*, 2016.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Generalizing Browser Extension CookieAudit for Auditing Consent Popups' GDPR Compliance

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Nastaly	Szymon Tomasz

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 26. August 2024	Szymon Nastaly

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.