PriSEC: A Privacy Settings Enforcement Controller

Rishabh Khandelwal[1], Thomas Linden[1], Hamza Harkous[2], and Kassem Fawaz[1]

[1]University of Wisconsin–Madison
[1]{rkhandelwal3, tlinden2, kfawaz}@wisc.edu
[2]Google Inc.
[2]harkous@google.com

Abstract

Online privacy settings aim to provide users with control over their data. However, in their current state, they suffer from usability and reachability issues. The recent push towards automatically analyzing privacy notices has not accompanied a similar effort for the more critical case of privacy settings. So far, the best efforts targeted the special case of making opt-out pages more reachable. In this work, we present PriSEC, a Privacy Settings Enforcement Controller that leverages machine learning techniques towards a new paradigm for automatically enforcing web privacy controls. PriSEC goes beyond finding the webpages with privacy settings to discovering fine-grained options, presenting them in a searchable, centralized interface, and – most importantly – enforcing them on-demand with minimal user intervention. We overcome the open nature of web development through novel algorithms that leverage the invariant behavior and rendering of webpages. We evaluate the performance of PriSEC to find that it precisely annotates the privacy controls for 94.3% of the control pages in our evaluation set. To demonstrate the usability of PriSEC, we conduct a user study with 148 participants. We show an average reduction of 3.75x in the time taken to adjust privacy settings compared to the baseline system.

1 Introduction

For decades, the "Notice and Choice" model has been the governing framework for disclosing and controlling online privacy practices [29]. Privacy *notices*, manifesting in lengthy privacy policies, inform users about how websites, devices, apps, or service providers handle their data. Online settings and menus provide users with options to opt-in for data collection, manage their communication and marketing preferences, and control the extent to which their data is shared. However, in their current forms, privacy control settings suffer from usability issues [10]. With the introduction of regulations like the GDPR [9] and the CCPA [34], online entities are required to provide more privacy control settings to the users. In some instances, these privacy controls have become more cumbersome to locate, often distributed across multiple pages. As we later show in this work, the users needed to navigate to ten URLs, on average, to adjust a privacy setting in our user study.

Taking Twitter as an example, to set their privacy preferences, the user should first expand the "More" side-bar menu, navigate to the "Settings and privacy" page, traverse the relevant settings tabs (each of which contains numerous submodules a user must enter), change the settings, and then exit. Additionally, the users already have to know the actual terms to look for when navigating these interfaces.

Therefore, users may find it hard to exercise informed privacy control for websites with deep menus for privacy settings. They are far more likely to rely on default configurations than they are to fine-tune their settings for each service [1, 15]. In several cases, these default settings are privacy-invasive and favor the service providers, which results in privacy risks [21, 24, 25]. While several proposals have aimed at alternative interfaces for presenting privacy notices [12, 30, 40], online privacy controls have received less attention. The main work in that context has been on automatically extracting opt-out links from privacy policies [11, 14, 32].

In this work, we propose a new paradigm to improve the accessibility of web privacy controls: we automatically find webpages with privacy settings, locate the fine-grained options within these pages, group them by topic, present them in a searchable user interface, and allow users to automatically enforce them on demand. Achieving these objectives requires (1) building a unified understanding of the privacy control settings that scales across providers and web technologies and (2) developing flexible user and programming interfaces that allow the user to interact with the settings in an intuitive way.

To realize these goals, we built PriSEC, a privacy settings enforcement controller that utilizes machine learning techniques to discover, present, and enforce privacy settings. To address the challenges described above, PriSEC leverages a key insight to enable the robust extraction of privacy control

elements: their presentation to the user and behavior should be consistent to maintain the user experience. Using this insight, PriSEC applies a three-stage pipeline that, given a domain, extracts a machine-readable representation of its privacy controls. First, PriSEC crawls the domain and identifies privacy control pages via a machine-learning classifier that exploits the site's textual and UI features. Next, PriSEC simulates users' behavior by interacting with every UI element on the page. Using a deep-learning-based visual classifier, it categorizes these elements into types, regardless of their underlying implementation. Finally, it clusters these UI elements into groups, creating "control recipes," ready to be consumed by the application interface.

To demonstrate these recipes, we built a Chrome browser extension that presents them to the users in a centralized location. Also, it enables the users to pose free-form natural-language queries that are semantically matched with the relevant privacy controls. Once the user provides their choices in the extension, PriSEC automatically enforces the relevant setting, without any further interaction, thereby making the privacy settings more accessible and reducing the overhead of the user at the same time.

We further perform an end-to-end evaluation of PriSEC, assessing its core components:

- We show that our pipeline for generating enforceable control recipes correctly extracts such recipes for 94.3% of the pages in our manually annotated control pages' dataset. This evaluation showcases the generality of PriSEC's design, despite the variance in the HTML implementation of the analyzed pages.
- We evaluate PriSEC's performance on matching user queries with privacy options, and we find that it achieves a top-3 accuracy of 95.6% on a dataset of free-form queries that originate from real users on Twitter and Reddit. This result shows the extent to which PriSEC can reduce the user's effort to locate a privacy setting of interest.
- We further conduct an online user study with 148 participants on Amazon MTurk to evaluate PriSEC's client implemented as a browser extension. We find that time taken to adjust privacy settings on a set of 6 popular websites is reduced by a factor of 3.75. Moreover, PriSEC received a higher average System Usability Scale (SUS) of 72 compared to 63 for the manual baseline.

2 Background on Privacy Settings

Before delving into PriSEC's design, we start with the necessary background and definitions around privacy settings that we use later in the paper. Online service providers offer privacy settings, in the form of *Privacy Control Pages*, for their

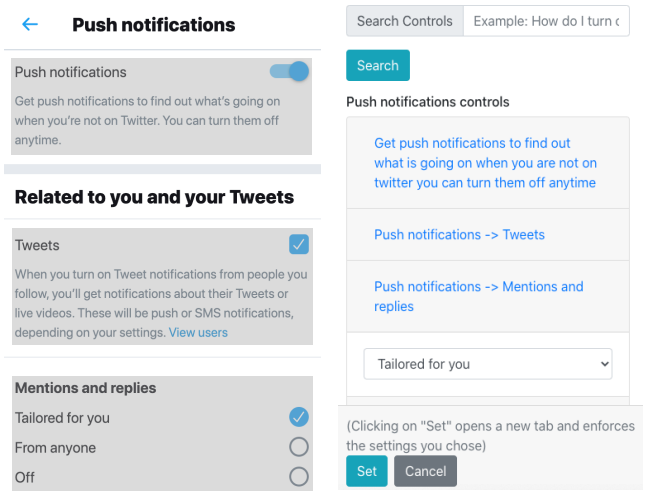


Figure 1: (Left) Settings page on twitter.com showing various groups. (Right) Rendering of the client side showing the search interface of PriSEC.

users to control the access, processing, and sharing of personal data. The anatomy of privacy control pages is typically different from other common types of webpages. They are not rich with text, and, depending on the domain, they might contain a set of input elements. Privacy settings can be either co-located within general settings pages or embedded in privacy policies. There are two types of privacy settings: browser-centric and user-centric. Browser-centric settings cover managing cookies through centralized user interfaces, such as browser settings or site-provided banners [35]. In contrast, user-centric settings require the user to find and interact with specific pages (sometimes requiring a login) designed by the service providers. We focus on user-centric settings due to their challenging, non-standard, and distributed aspects in this work.

One can view a privacy control page as a set of *Privacy Control Groups*. Each group is associated with a single privacy topic and a set of options for that topic. Fig. 1 shows an example of a privacy control page from twitter.com. The page contains three control groups corresponding to the topics: *Push Notifications*, *Tweets*, and *Mentions and replies*. For example, the *Mentions and Replies* group has three options: *Tailored for you*, *From anyone*, and *Off*. In our context, each privacy option is associated with an input HTML element with which the user can interact. This interaction results in setting a choice for that privacy topic. For instance, Fig. 1 shows a case where radio-button elements can be used to set one of three options for configuring mentions and replies. It also shows another case where a checkbox element can be used to enable push notifications for tweets.

In PriSEC, we use the term *Control Recipe* to refer to the sequence of actions required to set a specific privacy option. PriSEC utilizes a browser extension that presents these options in a centralized interface. The user can then decide on

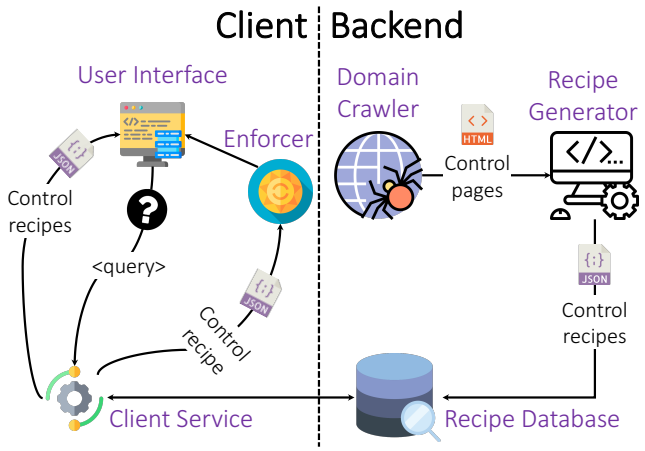


Figure 2: The system overview of PriSEC. The client-side handles the user interaction whereas the backend performs the offline processing and generates control recipes.

their *privacy preference* within the plugin. PriSEC then automatically enforces these preferences. To do so, it takes the control recipe of the privacy option and packs it as a JavaScript snippet. Then, it executes JavaScript in the corresponding privacy settings webpage.

3 System Overview

PriSEC extracts the privacy settings, presents them to the users in an intuitive way, and enables automatic enforcement of their choices. It employs two components: a backend component responsible for offline processing of control pages and building the control recipes, and a client-side JavaScript extension to handle user interaction. A high-level diagram of PriSEC is shown in Fig. 2.

Backend

Given a domain, PriSEC's *Domain Crawler* crawls its webpages to identify a set of privacy control pages. This crawler is described in Sec. 4. Then, the Recipe Generator processes each control page to extract a machine-readable representation of all the privacy control groups on the page (Sec. 5). We call such representations the *control recipes*. They include, for each control element, the XML Path (XPath) leading to it (representing the sequence of actions needed to set it) and a descriptive text extracted from the page.

Client

On the client side, PriSEC has a plugin running on the user's browser. As shown in Fig. 2, the *Client Service* locally maintains the per-domain control recipes that have been generated by the backend. Given a domain, the *User Interface* presents these recipes and allows the user to issue free form queries about specific control settings. To change a setting, the user can either locate the setting-of-interest through browsing the list of settings or issuing a free form query, as shown in Fig 1. The *Enforcer* module takes the user choices and automatically applies that setting in a new tab by injecting the necessary JavaScript, based on the control recipe, without any further user intervention.

In this paper, we present one example of a user interface that leverages the functionality of PriSEC. One could, however, use the underlying system as a more general-purpose API that can return and enforce control recipes as requested by a user-level interface.

Challenges

Designing and implementing PriSEC comes with a set of unique challenges. The major challenge stems from the open nature of the web domain. Unlike developing with mobile-device frameworks such as *Swift* or *Android Studio*, web development is far less structured. Further, the static resources of most modern webpages do not provide a comprehensive picture of the service, and the lack of uniform code structuring introduces additional nuances to any third-party analysis. For example, extracting relevant text for a privacy option is difficult because the implementation varies among websites. Thus, our goal of generalizing the search, processing, and enforcement across webpages is a challenging task.

The next sections explain the design of PriSEC's modules (Fig. 2), namely the crawler (Sec. 4), the recipe generator (Sec. 5), and the client application (Sec. 6).

4 Crawler

The *Crawler* module identifies the privacy control webpages for a target domain. This module takes a two-step approach: it first finds the candidate pages; then it classifies these candidates using a machine learning classifier that we developed. Fig. 3 highlights the operation of the crawler.

4.1 Candidate Page Identification

Given the domain, PriSEC's crawler finds a valid starting URL from the search results of DuckDuckGo, a popular search-engine. The search query is the domain along with the keywords: "privacy" and "settings". The crawler chooses the starting URL to be the first page with a domain that matches the target. Then this module extracts all visible anchor tags located at the starting page. We use the Selenium web-driver in Python to perform the crawling.

In certain domains, some anchor links are only visible upon clicking specific elements on the page (e.g., a profile icon). To reveal these hidden links, the crawler iteratively tabs (i.e., simulates a tab click on the keyboard) through the page and clicks all the interactable elements. The set of obtained links

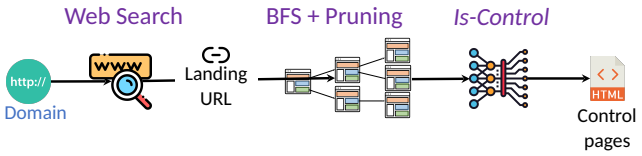


Figure 3: The processing pipeline of PriSEC's crawler.

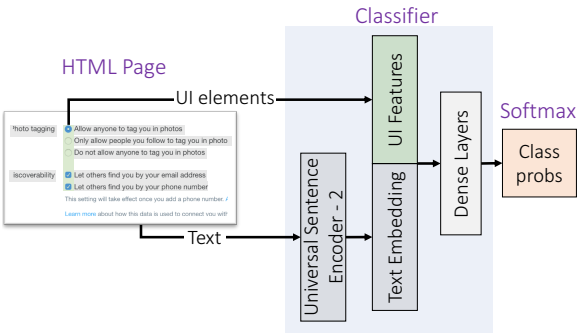


Figure 4: The architecture of PriSEC's *Is-Control* classifier.

are crawled in turn in a Breadth-First-Search (BFS) order with a depth of 3. In order to prune the search space, we apply a set of pattern-matching heuristics to eliminate URLs that are known not to contain privacy controls (e.g., `https://example.com/about` or `https://example.com/faqs`).

The resulting candidate set comprises the discovered anchor links during this process. We further apply a keyword-based filter targeting the page title to reduce the noise in this set. The keywords are derived from the set of privacy-control pages used for training the *Is-Control* classifier in Sec. 4.2.

We note that to find the candidate pages behind a login page, we first create Chrome profiles and manually login once to the websites either using third-party logins or test accounts. We then load these profiles in Selenium, which automatically reuses the sessions during crawling. We discuss the limitations of this step and the alternatives in Sec. 8.

4.2 *Is-Control* Classifier

Now that we have a candidate list of links, we want to keep the ones corresponding to control pages. PriSEC uses a custom *Is-Control* classifier that takes an input as a candidate HTML page (from the crawler), extracts its textual and visual features, and predicts whether that page is a control page.

4.2.1 Architecture

We consider two feature sets for representing an input HTML page: text features extracted from the page and visual features extracted from its UI elements. The model architecture is shown in Fig. 4.

Text Features To represent the page's text, we combine the page title, text from heading elements (`h#`), and text from the buttons. In the cases where the heading elements do not exist, we add the text from paragraph elements (`p`) if present, and from the entire text of the page if not. Next, we encode the combined text using a pre-trained Universal Sentence Encoder [5] based on a Deep Averaging Network [13]. This encoder has previously shown success for text classification tasks with small datasets [27].

Visual Features Then, the crawler enumerates all the input elements on the page and builds a binary feature vector. This 4-dimensional vector encodes whether each of the following types of UI elements is present or not: radio buttons, checkboxes, select elements, and buttons. The decision to make this feature vector a binary one was to restrict the input space, particularly because the training data is relatively small (as we explain later).

Combining the Features Concatenating the visual feature vector with the text embedding vector results in the input vector to a neural network composed of two dense layers with ReLU activation [2], followed by a Softmax. The classifier outputs a probability vector indicating whether the page is a control page or not.

High Recall Goal A major challenge for this classifier is the inherent noise in HTML pages, due to headers, footers, and side menus. This noise might affect the classifier's precision. However, our main goal from the *Is-Control* classifier is to act as an initial filtering stage. It should exhibit high recall on the control pages, capturing almost all control pages of a domain, but not necessarily high precision. The subsequent processing steps of PriSEC's pipeline will handle false positive instances.

4.2.2 Training and Testing

To create the data for the classifier, we used the privacy policies dataset from Linden et al. [18] as a starting point. Starting from the privacy policy links and the corresponding homepage, two of the authors went through the outgoing URLs, authenticating the user if necessary. In this process, we obtained 198 privacy control URLs (43 located behind logins) and 498 non-control URLs. In total, these sum up to 696 unique webpages. The non-control pages we select vary significantly in purpose as the overarching objective was to collect a diverse set of pages, including text-rich pages (*e.g.* articles), privacy-related pages (*e.g.* privacy policies), and pages containing forms (*e.g.* contact or login pages).

Next, we set aside a balanced test set of 100 pages split evenly between control and non-control pages. We train the binary classifier on the remaining set of 596 pages, with 148 control and 448 non-control instances. We used over-sampling during training to equally represent samples from the two

Table 1: A breakdown of *Is-Control* classifier's performance on the test set.

Instances	Support	Recall	Precision	F1-score
Control	50	0.98	0.84	0.91
Non-Control	50	0.84	0.98	0.89
Total Pages	100	0.90	0.91	0.90

classes. Table 1 shows the performance of the classifier on the test set. As evident from the table, the classifier detects 98% of the control pages and has a false positive rate of 16%. We purposefully optimized the classifier to have high recall over the control pages because the *Recipe-Generator* module is designed to further filter out the false positives, as discussed in Sec. 7.

5 Recipe Generator

The *Recipe-Generator* module receives potential privacy control pages from the *Crawler* module. It extracts a "machine-readable" and uniform representation of the privacy control page. This representation is the set of privacy control groups, the privacy options, and their control recipes, as defined in Sec. 2. With such representation in place, PriSEC enables a set of client applications that run on different HTML implementations of control pages. In Sec. 6, we provide an example of such an application.

In this context, we have to overcome two main challenges that arise due to the nature of web development. First, the growing popularity of dynamic-loading of webpage content suggests that static analysis of HTML is insufficient for discovering the HTML elements associated with privacy options. Second, in HTML, there is no standard way to implement the elements with which the user interacts. For example, *switches* can be implemented using *checkbox* as well as *div* elements. Identifying the type of an HTML element is important for recognizing the privacy options once discovered.

To overcome these challenges, we leverage a key property of webpages; regardless of their underlying implementation, they should render and behave consistently. PriSEC leverages the behavior and rendering invariants to extract the privacy control groups, options, and associated control recipes from the control pages. Fig. 5 shows a high-level overview of the *Recipe-Generator*'s processing pipeline; it assumes the following operation:

1. It mimics the user's behavior on a control page to discover all the accessible input elements and organize them in a graph structure (Sec. 5.1).

2. It uses the invariant rendering of the input elements to classify them into UI types using an image classifier (Sec. 5.2).

3. It identifies the privacy options from the recognized input elements and organizes them into privacy control groups. Then, it associates each privacy option with its control recipe (Sec. 5.3).

5.1 Extraction of Privacy Options

The *Recipe-Generator* begins by extracting the set of all candidate privacy options within a control page. This process involves identifying candidate options and organizing them in a dependency graph, which is later used to generate the control recipes.

Discovering the Initial Set of Focusable Elements: PriSEC's identifies candidate options through discovering interactive elements on a webpage. Regardless of their underlying HTML implementation, privacy options represent elements with which the users can interact. PriSEC's *Recipe-Generator* module leverages the fact that, by default, all components designed to handle user interactions are expected to be focusable. Originally introduced to increase the accessibility of webpages, focusing allows browsers to designate which interactive element on a page currently receives keyboard inputs. Users can switch between elements by pressing the TAB key (an action to which we refer as *tabbing*).

The *Recipe-Generator* module simulates a tabbing behavior to identify all the focusable elements on a page. To reduce the amount of noise in this focusable set, PriSEC processes a second webpage from the same domain and filters out all the focusable elements that are common to both pages. This reduction technique primarily targets the headers, footers, and side-navigation walls of webpages, which tend to render similarly between pages in the same domain. Our underlying assumption is that privacy options are unique to the control page, unlike other input elements.

Click Analysis: The process we described so far misses the HTML elements which are dynamically injected or enabled. A typical example of control elements that are dynamically injected is on the left side of Fig. 6. In this example, the privacy options controlling the push notification types become visible only after the switch button is enabled. To capture such elements, PriSEC performs click analysis by simulating the user clicks on the identified focusable elements. Each action might result in dynamically injected HTML code, which is then analyzed using the same tabbing approach.

The other types of elements that the previous process misses are disabled elements, such as the ones on the right side of Fig. 6. Before clicking the "switch" element in the top right corner, the "checkbox" component was disabled and inaccessible. The *Recipe-Generator* tests how the focusable elements react in response to click actions to identify the accessible elements as well the interaction sequence that leads to

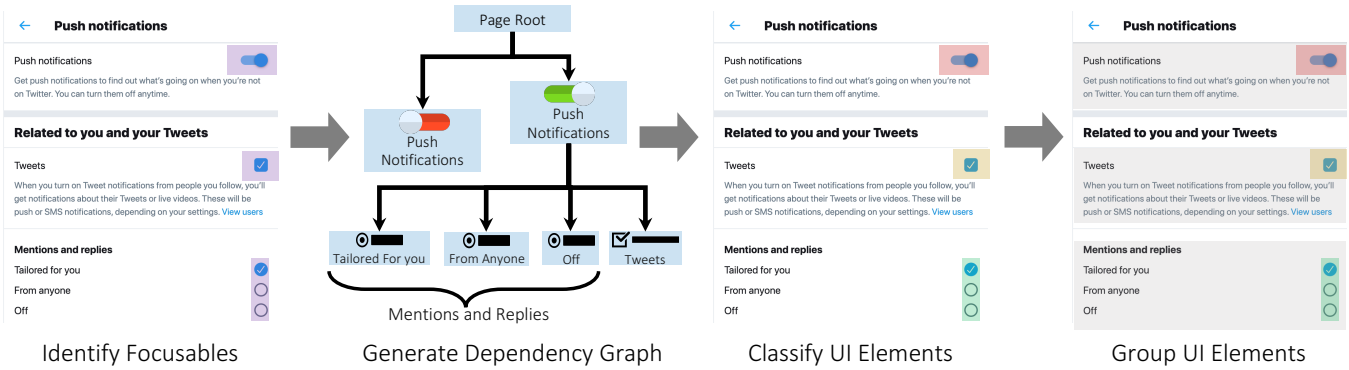


Figure 5: The recipe generation pipeline of PriSEC. It starts by identifying the focusable elements on the page. Next, it generates the dependency graph, which captures the relations between the elements, if any. Then, it classifies the elements according to their UI type using a visual classifier. Finally, it groups the UI elements in control groups. These groups are used to build a control recipe representing the privacy choices and controls of a page.

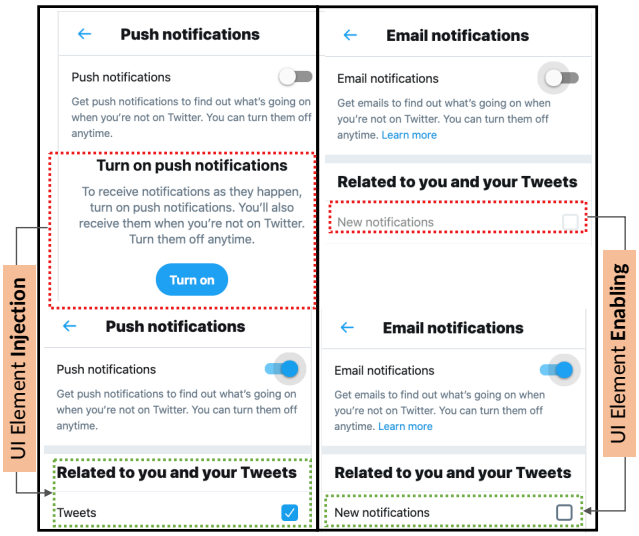


Figure 6: Examples from the PriSEC's click analysis. One highlighting the discovering dynamically injected options, the other showing dynamically enabled options.

each element. The final set of candidate options includes both the original and dynamically discovered focusable elements. We refer to this set of candidates as the "*focusable-set*".

Constructing a Dependency Graph: PriSEC not only detects option candidates; it also extracts the sequences of actions required to reach each focusable element. PriSEC's client applications can utilize these sequences to enforce user privacy preferences. Accordingly, the *Recipe-Generator* organizes the focusable-set into a directed acyclic graph (second step in Fig. 5). The graph nodes represent focusable elements, and directed edges between the nodes represent the destination node's dependency on the source node's execution. As such, the leaves in this graph represent the privacy options.

To construct this graph, we use the interaction sequences that lead to each focusable element. First, we define a placeholder source node as the initial state of the webpage. We then create a set of edges from that source node to the focusables which are accessible upon that initial page loading. Then the *Recipe-Generator* clicks on each focusable to discover the dynamically injected and enabled focusables. Whenever a click on a focusable reveals another previously unseen focusable, the *Recipe-Generator* creates an edge between a copy of the source and destination elements. As such, the source focusable will appear twice in the graph: as a leaf in the graph as well as a parent to the newly discovered nodes (e.g., "Push Notifications" in Fig. 5). The process continues in a depth-first manner until no new elements are discovered.

5.2 UI-Element Classification

After discovering the candidate elements for privacy options, the *Recipe-Generator* module identifies their types, which is important for automatic enforcement. For example, a user can only choose one radio button from a group but can check several checkboxes in a control group. PriSEC classifies the visualized rendering of candidates as one of seven possible types of UI focusable elements ("text", "button", "link", "radio button", "checkbox", "switch", and "select").

Empirically, we found that the HTML attributes of "radio buttons" and "text inputs" were consistent and reliable across the top 500 websites from the Amazon Alexa Top Sites List, which was not the case for the rest of the UI element types. For example, we found that "switches", "buttons" and "selects" can all be implemented using the "div" element, making it impossible to classify them only based on the HTML tags. Thus, PriSEC first leverages the HTML of a webpage to deterministically identify "radio button" and "text inputs" elements.

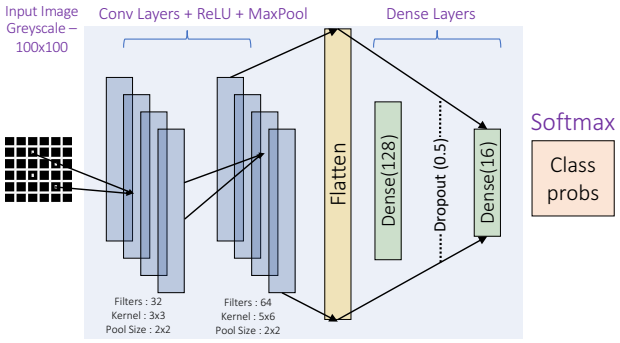


Figure 7: The architecture of the visual classifiers for PriSEC.

For the remaining five types of elements ("button", "link", "checkbox", "switch", and "select"), we designed a classifier to identify the type of focusable element using its screenshot. The screenshot of the element is automatically taken using its coordinates and the full control page's screenshot.

UI Component Dataset Usually, training such visual classifiers requires a large amount of labeled data. From the top 500 websites from the Amazon Alexa Top Sites List, we observe that many privacy control pages exhibit just one or two controls. Instead of manually labeling the elements from these control pages, we create a synthetic dataset of UI components that we can easily scale and introduces a wider range of variations in style and size.

We implemented a *ReactJS* web application loaded with 11 popular React UI building libraries (listed in Appendix A) to generate the synthetic data. We traverse through the libraries' implementations for each of the five UI component types and render a UI component for each available style the library offers. For introducing further variations, we populate the text-containing elements (the default label for selects, the anchor links' text, and the button text) and render a component instance for each variation. Finally, for the binary-state components (*checkboxes, and switches*), we render an instance for both the checked and unchecked states. Our final dataset has 699 UI components, composed of 450 *buttons*, 13 *switches*, 100 *links*, 28 *checkboxes*, and 108 *selects*.

Visual Classifier: The visual classifier is a convolutional neural network consisting of two convolutional layers for feature extraction followed by two dense layers for classification. A schematic diagram of the architecture is shown in Fig. 7. We further augment the synthetic set using horizontal and vertical shifts to emulate the irregularity in screenshot capturing. We then train the classifier on this augmented set by splitting data into two sets: train (80%) and test (20%); we used early stopping to prevent overfitting. The classifier has a near-perfect accuracy (around 99.9%) on the five classes on the synthetic set. We evaluated its performance on 102 UI elements extracted from control pages of the top 500 websites from the Amazon Alexa Top Sites List. Table 2 shows the classifier's F1-score for each of the five classes. The classifier generalizes well on the samples from the wild (average F1-score of 98%), despite being trained on synthetic data.

Table 2: Performance of the visual classifier

Class	Precision	Recall	F1-score	#(Elements)
checkbox	1.00	1.00	1.00	26
select	0.93	0.93	0.93	14
switch	1.00	1.00	1.00	26
button	0.95	1.00	0.97	18
link	1.00	0.94	0.97	18
Total	0.98	0.98	0.98	102

5.3 Constructing Control Recipes

With each node (an input element) in the directed graph now tagged with its corresponding UI type, the *Recipe-Generator* seeks to group options representing a single privacy topic. As a first step, it distinguishes between different roles that these elements can play: "selectors" that represent privacy options (such as checkboxes) and "enforcers" that are used to apply these options (such as a "save" button). Then, the elements are assembled to create a machine-readable representation of the privacy control page: a set of control groups, each associated with text describing the privacy topic (*per-group* text). A control group has a set of privacy options, each represented with its own text (*per-option* text). Then, the *Recipe-Generator* associates each privacy option with a control recipe.

5.3.1 Selector vs. Enforcer Tagging

PriSEC defines two execution roles for candidate options. "Radio buttons", "check-boxes", "text-inputs", "switches", and "selects" are categorized as *selectors*; the role of these components is choosing privacy options. However, if a user were to solely interact with these components, in many cases, their choices are not submitted. In these cases, the control page has some "button" or "link" that acts as the *enforcer*. The *Recipe-Generator* first categorizes the components into their execution roles based on the detected element type. Should an enforcer exist on the page, this module associates the *selectors* with that *enforcer*. In the case of multiple enforcers, we choose the enforcer closest to a given selector by comparing on-screen distances between the selector and the enforcers.

5.3.2 Privacy Option Grouping

The next step of the *Recipe-Generator* is to build the control groups from the identified selectors. First, it sorts the selectors according to their order of appearance in the HTML of the control page. Then, it forms the group using this guiding

principle: each group is the list of consecutive sel
the same UI type that share the lowest common an
HTML parse tree. For example, the slider in Fig. 5 v
one group, the checkbox will form the second group
radio buttons will form the third group.

To provide context for the privacy options, the
Generator module extracts the relevant text for the
(per-option text) and the groups (per-group text). Fo
option text of the selector, the *Recipe-Generator* sea
the closest node in the HTML parse tree, which cont
This node can be the selector element itself, an ance
child of an ancestor. The *Recipe-Generator* forces tv
tions for this node: it should not have any other sele
child, and its text should not have been used for anot
tor. Both conditions ensure that the per-option text i
to the selector. For the *per-group text*, the *Recipe-G*
searches for the lowest common ancestor of all its s
which contains text. The only condition is that the te
not be the per-option text of any of the selector elen

5.3.3 Privacy Control Recipes

Finally, PriSEC generates the control recipe for eacl
option. Recall that each privacy option is represented
selector in the dependency graph (second diagram i
Also, each selector either acts as an enforcer or is assigned a separate enforcer element. The control recipe is the path from the root of the graph to the enforcer of each privacy option. The path includes the list of UI elements required to reach the privacy option.

PriSEC leverages XML Path (XPath) queries to implement the privacy control recipes. XPath expresses the location of an element via a query starting at an anchor point in the page (an element with a fixed HTML attribute). The client-side scripts of PriSEC use the XPath expression of each element in the recipe to automatically locate it and perform user actions such as clicking. The sequence of these actions allows PriSEC to reach the privacy options and set the user's chosen value.

6 Client Application

As discussed in Sec. 3, the client application of PriSEC is a browser extension supported by a natural language query interpreter. PriSEC's client presents the users with an interface to view and search for the privacy options. The interface, as shown in Fig. 1, is designed to serve two purposes: the viewing option allows the users to learn about the privacy settings offered by the given website, and the search option allows the users to search for their preferences. Using this interface, users can decide on their preferences and interact with the extension to enforce them in an automated way. In this section, we discuss the components and the workflow of the client application.

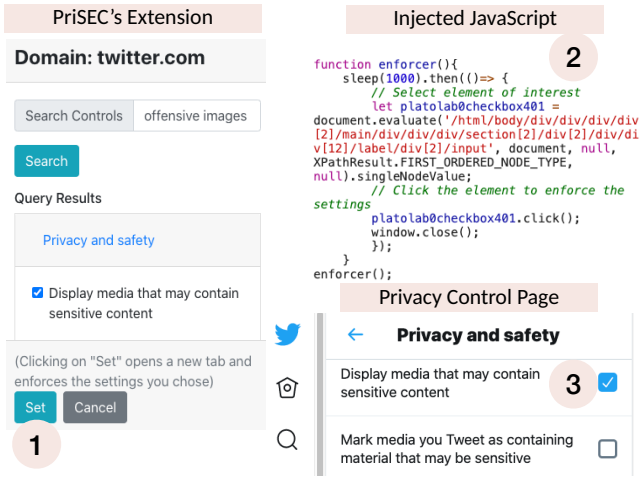


Figure 8: A typical workflow of enforcement in PriSEC: 1) User searches for a setting and provides their choice. 2) PriSEC generates the JavaScript for enforcement and injects it in a new tab. 3) Status of the setting after enforcement

.

6.1 User Interface

The user can activate PriSEC's browser extension by clicking the extension icon in their browser. The extension consists of a basic form interface that renders the list of privacy options that PriSEC identified. For those domains providing numerous privacy settings, users might find it cumbersome to navigate all the options and select the ones matching their preferences. PriSEC improves the accessibility of the privacy options by including the ability to semantically search for relevant privacy options, as shown in Step 1 of Fig. 8.

As previously depicted in Fig. 2, the Client Service (explained below) handles the user's search queries. Then the interface presents the matching privacy control groups in a sorted order according to their semantic similarity to the user's query.

6.2 Client Service

The client service is responsible for two main tasks (as depicted in Figure 2): (1) managing the life-cycle of control recipes and (2) performing the semantic matching of the privacy options with user queries. Starting with the first task: for a given website, the client service fetches the control recipes and exposes them to the user interface. In our implementation, the service fetches these recipes on-demand from the backend. In case the extension is required to operate without communicating with the backend (e.g., if the client prefers not to send the timestamped URLs it visits), PriSEC can package the recipes within the extension and update them periodically for common domains. Once the user makes a choice in the extension's user interface, the corresponding control recipe is

pushed to the enforcer module.

The second component of the client service performs the semantic matching. This component encodes the user's query text and the text associated with each privacy option in the same embedding space. Then, it ranks the control groups according to the cosine similarity between their embeddings and the query text. We investigated several types of pre-trained encoders that target the task of semantic similarity, including Universal Sentence Encoders [5] and SBERT [31]. We evaluate several variants of this approach in Sec. 7.3. Additionally, we compare large models designed to work on the server and small models that function completely within the browser extension. Our goal is to understand the trade-off between higher accuracy (larger models) and better privacy (locally resolving user queries).

It is important to emphasize that the recipes are only fetched when the user clicks on the browser extension. Further, the time cost of loading the recipes and performing a search is relatively small: it takes around 100ms to load the recipe and 300ms to execute a semantic search query.

6.3 Enforcer Module

PriSEC triggers the enforcement when the user selects a privacy option and clicks on "set". The module first retrieves the recipe associated with that option and dynamically generates the JavaScript code for executing the recipe. Leveraging the elevated privileges granted to extension, the application opens a browser tab and injects the corresponding enforcement JavaScript code. Fig. 8 shows the dynamically generated script (Step 2) and the adjusted privacy option (Step 3) as a result of the user action. Throughout this process, the user only interacts with the popup screen of the browser extension – PriSEC sets the privacy preference automatically without the user's involvement. Once done, the user can navigate outside that popup and continue the regular website experience. A working demo of PriSEC can be found by navigating to the links in the footnotes below[1,2].

7 Experiments

We perform an end-to-end evaluation of PriSEC, evaluating its core components. The evaluation covers the complete pipeline of PriSEC: the crawler, recipe generator, and client applications. The experiments corresponding to the evaluation of our system are as follows:

Experiment 1 — End-to-End Evaluation: We evaluate the automatic identification and annotation of privacy control pages by testing the *Crawler* and *Recipe-Generator* modules against a set of 100 privacy control pages never before seen by our system. We compare the results to a manual extraction executed by the authors.

Experiment 2 — Semantic Matching: We test the natural language query interpreter against a set of relevant user queries asked on Twitter and Reddit about the domains in our privacy control set. We compare the results to the ground truth annotated by the authors.

Experiment 3 — User Study: We conduct an online user study with six popular websites to evaluate the usability of the automatic presentation and enforcement modules of PriSEC.

7.1 Datasets

We curated two new datasets to evaluate PriSEC. The first consists of a set of control pages from popular domains, and the other consists of questions people posted about privacy settings on Twitter and Reddit. The development of PriSEC was entirely blind for these sets, which we curated solely for evaluation purposes.

7.1.1 Privacy Control Pages (PCP) Dataset

We manually curated an evaluation set from the top 500 websites from the Amazon Alexa Top Sites List[3]. For each domain, two authors manually searched for the privacy control pages. We created alias accounts for those websites requiring logging in and navigated the user settings to ensure that all control pages are captured. After removing the non-English pages and discarding the domains we had already seen in training/testing sets of the *Is-Control* classifier, we were left with 100 privacy control pages across 58 unique domains.

7.1.2 Natural Language Queries (NLQ) Dataset

To evaluate PriSEC's search-based interface with realistic questions, we created a new query *test* dataset covering privacy settings. We developed this dataset with two goals: (1) including free-form queries around the privacy settings of the domains in our PCP dataset, and (2) ensuring that the queries correspond to existing privacy options within the control pages. To achieve these goals, we collected questions from Twitter and Reddit that users had asked about the privacy settings of domains in our PCP dataset. Following this approach avoids the biases related to soliciting questions from individuals about privacy options [12]. Consistent with research on evaluating human-annotated queries [7, 23, 36, 37], we sought a dataset with a size in the range of 100-200 queries.

For extracting queries from Twitter, we followed a methodology inspired by Harkous et al. [12]. We searched for reply tweets that contain the URLs of the pages in our PCP dataset. Then, we backtracked each reply to get the original

[1] Reddit: https://youtu.be/Am27HdQ5u1w
[2] Twitter: https://youtu.be/YXHwPGg_Z-M

[3] https://www.alexa.com/topsites

tweet, which includes the question that solicited the reply. We automatically filtered the resulting queries to keep those containing question marks and at least four words, resulting in 77 tweets containing free form queries by the users.

We followed a similar methodology for Reddit, and we searched within threads located in subreddits corresponding to the domains in our PCP dataset. This process resulted in 101 queries. An example from the Twitter subreddit is: "Anyone know how to make it so people gotta request to follow and see my tweets?"

At this point, we have automatically collected a set of 178 candidate queries about privacy settings from Twitter and Reddit. These candidates constitute a superset of free-form queries that address privacy controls. Next, two authors independently analyzed each query and decided whether it is a valid query about some privacy control. The authors manually tagged each query with the control element that answered the query – discarding the queries without an answer. The annotators exhibited a near-perfect agreement on the annotations of the queries (valid vs. invalid) and the answers to the queries. In particular, Cohen's Kappa for both authors was very high ($\kappa = 0.82$) [16]. They both tagged 122 of the queries as valid and 43 as invalid. They only disagreed on the answers for 13 queries, which they resolved after discussions. The final outcome of this process is a set of 135 queries covering 15 domains, including the answer to each query.

7.2 End-to-End Evaluation of the Backend

We perform an end-to-end evaluation of PriSEC starting with the 58 unique domains of our PCP dataset. Our objective is to extract the machine readable representations of the control pages across these domains and manually assess their correctness. This evaluation includes validating the control group, the per-group text, the privacy options, the per-option text, and the privacy control recipes.

We first pass the domains of the PCP dataset as arguments to the *Crawler* module which returns a total of 9909 candidate URLs for control pages, with the mean number of extracted URLs per domain being 170. The keyword-based filter of the crawler (Sec. 4.1) reduces the number of candidates to 1400. This set of candidates contained 95 of the 100 URLs for privacy control pages of the PCP dataset. The webpages that the *Crawler* missed implement their navigation to privacy control pages without hyperlinks, which are currently out of scope. Analyzing these pages requires computationally demanding interactions with the websites which slows the crawling.

Next, the *Is-Control* classifier classifies 323 pages out of the 1400 candidates as control pages ("positive" label). This set contains all the remaining 95 control pages from the PCP dataset, indicating that the recall of the classifier on this set is 100%. Manually analyzing the remaining 228 pages classified positively, we find that: (1) 29 of these pages are present in our PCP dataset but with a different URL, (2) 23 of them are privacy policy pages, (3) 59 of them are settings pages which do not contain privacy settings, and (4) 107 of them are pages which have privacy related content (like blog posts) but are not privacy control pages. Further, we find 10 new privacy control pages which were missed during the manual annotation.

Table 3: Details of the URLs that were missed in Recipe Generation

Domain	Pages Missed / Total Pages In Domain	Total Groups	Num Groups Missed	Comments
Goodreads	1/3	21	15	Incorrect Text Extraction
Medium	1/1	8	6	Radio elements implemented as buttons
Mediafire	1/1	1	1	Not Reachable using *tabbing*
Daily Mail	1/1	10	-	Enforcers not captured; nested in tables
Wordpress	2/5	9	9	Tables implemented using *div* elements

At this stage, we have 323 pages tagged as privacy control pages by the *Is-Control* classifier, out of which 105 are actual privacy control pages. Next, the *Recipe-Generator* module processes this set of pages; we evaluate the annotation, grouping, and recipe generation of the *Recipe-Generator*. We find that the *Recipe-Generator* is able to correctly extract recipes for 94.3% of the actual privacy control pages. We further analyze the pages that the *Recipe-Generator* missed manually. The summary of this manual analysis is shown in Table 3. We observe that the *Recipe-Generator* misses the instances where the HTML implementation deviates significantly from the standard web practices. For example, the *Recipe-Generator* cannot analyze tables that do not use the HTML `<table>` tag. This result is not surprising because the hierarchy of HTML elements inside custom tables differs from that of a normal control page. We note here that while determining the number of pages missed (Table 3), we take the conservative approach and tag a page as missed if it contains any errors (missing group, missing option or extracting the wrong text). Further, we note that other than non-standard HTML implementation, we did not find any underlying pattern in the type of pages missed by the *Recipe-Generator* module.

For the remaining 218 pages, the *Recipe-Generator* module only generates recipes for 54 pages. These pages refer the users to general settings pages and contain control elements. The rest of the pages (164) are filtered out as they lack any unique control elements. Effectively, the *Recipe-Generator* acts as a second stage filter for the false positives from the *Is-Control* classifier. While these generated recipes are not privacy related per-se, they do not have a significant impact on the user experience. This is particularly the case for users who

utilize the semantic matching component of PriSEC, where they issue specific queries that sort privacy control groups based on their similarity to that query.

The main takeaway of this evaluation is that despite the variance in the HTML implementation of the analyzed pages, PriSEC accurately annotates 94.3% of the control pages.

7.3 Semantic Matching

We use the NLQ dataset to evaluate the natural language query interpreter in PriSEC's client. This set contains 135 questions about 15 domains from our PCP dataset. In our dataset, we observe that the average number of privacy control groups per domain is 11. We pass each query to the semantic matching module alongside the automatically collected privacy options for its domain. The ground truth for these queries was determined by the independent manual annotation by two authors as part of the NLQ dataset.

We evaluate the performance of this module using several encoders. The Universal Sentence Encoder (USE) [5] encoder is trained with a Deep Averaging Network (USE-DAN) [13]. In addition to the full model (916 MB), we include the lightweight version USE-Lite (25MB). We also include two other encoders, based on SBERT [31] and SRoBERTa [31], which are finetuned versions of BERT [8] and RoBERTa [22] using siamese and triplet network structures. These models are first trained on Natural Language Inference (NLI) datasets, then fine-tuned on the Semantic Textual Similarity dataset (STSB).

The results from the evaluation are compared in Table 4, which shows that the USE model outperforms the other encoders in this task. The results indicate that the user now only needs to see the top 3 control groups 96% of the time on a domain, as compared to browsing around 11 control groups, on average. Even if the relevant control is not found in the top 3 results, the user is almost certain to find the relevant group in the top 7 results. A near perfect top-7 accuracy is particularly useful for websites like Twitter for which 24 control groups were extracted. These results further confirm that the semantic matching component can play an important role in reducing the user's burden, making it easier to enforce their privacy preferences.

The difference between the accuracy of USE and USE-Lite is around 3.8% for the top-1 accuracy. Hence, it is possible to keep the semantic matching component completely on the client-side while partially sacrificing the matching accuracy. From a timing perspective, the local query with USE-Lite took 100ms on average on a Macbook Pro 2017 model. Guided by this result, it is possible to make an informed decision at deployment time concerning the privacy-utility trade-off in PriSEC.

Table 4: Top-k accuracy in % for the different encoders in semantic matching on the NLQ dataset

Model	Top-1	Top-3	Top-5	Top-7
USE	66.7	89.6	95.6	100
USE-Lite	62.9	84.4	91.8	97.8
SBERT-nli-stsb-base	48.2	76.3	84.4	92.6
SRoBERTa-nli-stsb-base	45.9	69.6	85.2	90.4

7.4 User-based Evaluation

We perform a user-based evaluation of the PriSEC extension through recruiting 148 participants from Amazon Mechanical Turk. We chose participants with $>$ 90% HIT approval rate who reside in the United States. The location criterion ensures that the participants were familiar with the test websites and their services. We paid each participant $4.00 to complete the study that lasted 21 minutes on average. Out of all the participants, 69% were male, 30% were female, 64% had at least a Bachelor's degree, and 32% did not have a degree. The average age of the participants falls in the age range of 25-44 years. We did not ask for any personally identifiable information, and the IRB at our institution approved the study.

7.4.1 Study Design

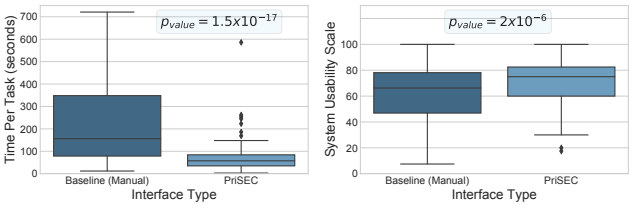
We develop a within-subject user study to assess the usability of the PriSEC extension. We used limited deception in that we did not expose the study's purpose to be about improving privacy settings' interfaces. In the study, we ask the participants to perform several tasks on a set of six websites. These tasks are derived from the queries of the NLQ dataset that we described in Sec. 7.1. The wording of the tasks reuses the queries themselves, with a few changes to address the task to the user. An example task for *Twitter.com* is shown below:

Query : Does anyone know if there is a ways how you can set that people you don't follow back can still DM you?

Task : You would like to set that people you don't follow back can still DM you. Find the corresponding setting and change it.

For this study, we choose six popular websites: `amazon.com`, `duckduckgo.com`, `twitter.com`, `reddit.com`, `flickr.com`, and `spotify.com`. We generated three tasks for Amazon and DuckDuckGo as there were very few queries for them in the dataset. For the other websites, we generated five tasks each, resulting in a total of 26 tasks.

In the study, the participants first install the PriSEC extension from the Google Chrome Web store. Then, we ask them to select the websites they are familiar with from the six websites. For each participant, we randomly select two of the selected websites and assign a task to them. This way, each user performs two tasks, corresponding to the baseline condition and the PriSEC condition. The baseline condition



(a) Time per task (b) System Usability Scale

Figure 9: The results of our user study suggest a decrease in time per task and an increase in usability using PriSEC versus baseline.

involves manually searching through the website to achieve the task's goal. The PriSEC condition involves using the PriSEC extension to perform the task. To account for the fatigue effect, the order of these two conditions is randomized per user. We ensure that, for a given user, the two tasks are for different websites to avoid any learning effects. For websites where the privacy settings are behind a login, the participants are instructed to log in before starting the task. We include snapshots of the tasks in Appendix B.

We measure the user effort by recording the total time each participant takes to complete each task. To calculate the time, we ask the participants to start from a fixed page (`https://example.com`) and use this as an anchor to determine the start time. We also store the URLs that they visit (on the website of interest) during this time. At the end of each task, the users fill the System Usability Scale questionnaire [4]. After each task, there are several checks in place to ensure that the participants have finished the task. We conclude the study with an open-ended question asking for general feedback about the extension. The final study was a result of an iterative process which included several pilot runs on Amazon Mechanical Turk.

To ensure that no harm was done to participants due to the study, we asked them to go back to the privacy control page (by providing them with the URL of the page where they changed the setting) and to adjust the settings according to their preferences. That way, we partially mitigate the risk associated with asking the participants to change their privacy settings. Still, the effect cannot be completely eliminated if some data was shared due to the temporary settings during the study.

7.4.2 Findings

Fig. 9a compares the average time the participants took to complete the tasks using the PriSEC extension and the baseline system (manually searching the website). The PriSEC extension performs better than the baseline method. On average, the participants took 3.75x more time to complete the same task when using the baseline method. To test the statistical significance of the result, we perform the Wilcoxon Signed rank test [39] as the data is not normally distributed. We reject the null hypothesis that the difference in time taken is not significant with a *p-value* of 1.5e-17. As a second indicator for user effort, we find that, on average, the participants visited 10 URLs before finishing the task using the baseline method. In PriSEC, however, the user can find and change the setting in just a few clicks within the extension without getting their browsing session interrupted. This result shows that PriSEC reduces the user's effort and time for configuring privacy settings.

Table 5: Analysis for user effort (average time) and usability (SUS score) for each domain used in the study. The entries with * denote that the change is statistically significant after accounting for multiple hypothesis correction ($p < \frac{0.05}{7}$)

Website	Avg Time (sec)		Avg SUS Score		# Participants	
	Manual	PriSEC	Manual	PriSEC	Manual	PriSEC
Amazon	348.9	**53.8***	72.5	**73.5**	32	42
DuckDuckGo	185.5	**39***	58.7	**74.8***	17	14
Flickr	501	**54.3***	65.6	**66.3**	4	6
Reddit	237	**94.6***	60.2	**66.5**	48	32
Spotify	181.3	**71.3***	64.5	**74.7**	20	30
Twitter	282.9	**90.7***	59.8	**72.9***	27	24

Next, we evaluate the usability by comparing the SUS scores in Fig. 9b. SUS scores are widely used in the literature [3] to compare different UI designs; a SUS score of more than 68 is considered above average [33]. With an average SUS score of 72, PriSEC again outperforms the manual baseline (average SUS score of 63), and we reject the null hypothesis with a *p-value* of 2e-6.

Table 5 shows the breakdown of average time taken and SUS scores with the websites that we used in the study. For the average time taken, it is evident that PriSEC performs significantly better than the manual method. The average time taken for manual tasks on DuckDuckGo and Spotify is lower than the others, indicating that the participants found it easier to locate the settings on these websites. Comparing the average SUS score, we see that PriSEC obtains a higher score for each website. However, the change is not significant (after accounting for multiple hypothesis correction) in websites like Flickr, Amazon, Reddit, and Spotify.

It is important to note that the participants are interacting with the extension for the first time, which reflects in the average time taken to complete the task. In many cases, the user tried to set a couple of extra settings to test and understand the extension. Furthermore, since the users were not aware that these were timed tasks, they may have taken breaks between completing the tasks. Our within-subjects study design accounts for this effect, which is common to both the tasks.

The majority of the participants who responded to the open-ended feedback question exhibited a positive sentiment towards the extension. A couple of the comments from the users are: *"... I love how I won't have to learn a new system every*

time I want to change a setting and I can just search the same way every time…" and *"…seems to make it easier to find options that may be buried behind multiple clicks …"*. More comments are listed in Appendix C.

8 Discussion

This section describes the potential technical limitations of PriSEC, touching upon the deployment aspects, and suggesting further extensions.

Limitations. The majority of limitations for PriSEC derive from the high variance in web technology implementations. For example, websites might require users to fill multiple text inputs before pressing an enforcer element (e.g., a button). While PriSEC presents these controls, handling multiple selectors before enforcement is out of scope.

Further, PriSEC works in the general case scenario of web implementation, but, as evident from Sec. 7.2, there are a few cases where the system fails. In principle, there can be two types of failures: a) recipe generation failures and b) enforcement failures.

We analyzed an additional set of privacy control pages to uncover possible patterns of error in the recipe generation. We tested the *Recipe-Generator* on a set of 55 privacy control pages from an additional set of 40 domains not used in development (or the evaluation). We extracted these domains from Linden et al. [18]. We found that the results are similar to what we observed in Table. 3. The *Recipe-Generator* module missed three pages. Two of the pages missed were due to non-standard HTML implementation ('anchor links' used for 'selects') while one page was missed due to the group text extraction failure. Some of these recipe generation failures can be detected by relying on user feedback, which can trigger manual reviews.

On the other hand, failures in enforcement result from stale recipes (due to the evolving site's HTML). These failures can be detected locally by checking the errors in the extension. Upon detection, PriSEC can trigger a recipe update for that webpage on its backend.

Another limitation of PriSEC is when its backend cannot log in to the website. We mitigate this issue by relying on third-party social logins (such as Google, Facebook, and Apple) existing in websites. We provide PriSEC's backend with test accounts on major social login providers to discover their recipes. We have a human-in-the-loop fallback for the remaining cases, where the system maintainers create the necessary logins.

The evolving nature of webpages can also cause the recipes to refer to stale elements. It is possible to mitigate this issue by replacing the fixed 24-hr period of recipe generation with a learned, dynamic period that accounts for the size of changes seen with time.

Furthermore, PriSEC shows its users the potential privacy options without considering their existing settings. Keeping the extension aware of these settings is challenging for any entity that is not the service provider. Hence, we accept this as a potential limitation in the user experience. We also note that studies aimed at understanding the effect of PriSEC on user choices are left for future work. Similarly, we have only considered a non-adversarial context in this work; studying how the system would operate in an adversarial scenario is also left for future work.

Deployment Aspects. PriSEC seeks to alleviate the user burden of enforcing privacy preferences. The system is intended to be used as an assistant while interacting with privacy settings. Hence, it builds on top of the existing choices offered by domains; it does not seek to replace them. Given that PriSEC increases the usability of the domain's privacy controls, we believe that there is an incentive for sites to encourage its use. For instance, PriSEC can be deployed in a guided mode to reduce the potential implications of an automated solution. In such a mode, users can see how to enforce their privacy preferences in a step-by-step fashion. This mode can also reduce the concerns about the non-perfect aspects of machine-learning-driven solutions to enforcing privacy preferences. Unlike most other privacy-conscious applications, our system works *within* regular user workflows. PriSEC imitates a user by opening tabs in the browser, navigating to the control page URLs, and sending user actions to the appropriate components. As more choices become available (e.g., due to the emergence of new regulations, such as the GDPR or CCPA), PriSEC can provide users with the newly available options.

Further Applications. One can view PriSEC an extensible framework that takes a domain and returns control recipes that are automatically enforceable. We provide a sample application that builds on top of it. We envision further extensions where the user declares a set of preferences within PriSEC; these preferences can be automatically enforced/suggested for new websites, akin to the proposed approaches for Android permissions [26, 38]. While Android permissions are standardized, online privacy settings are not.

PriSEC further leverages the semantic similarity encoders to match the users' preferences with privacy settings across websites. Using the semantic similarity, it is also possible to group similar settings to enable users to set particular preferences for all websites, instead of asking them to set preferences for each website. This approach would require extending PriSEC to support matching user queries with the privacy options.

9 Related Work

Automating settings in mobile context: There exists a long thread of research on automating the configuration of permissions on mobile operating systems [17, 19, 20, 26, 38]. Liu et al. [20] studied the feasibility of constructing generalized privacy profiles that predict user permission decisions. Further followup works also conducted field studies with actual users to test the usability of such profiles [19]. Wijesekera et al. [38] and Olejnik et al. [26] designed systems for dynamically granting user permissions based on users' preferences or context. When it comes to privacy settings, Chen et al. [6] were recently the first to study the discoverability of these settings for Android applications systematically. Their methodology uses static analysis to extract the elements within the UI layout. It then leverages the semantic relationship between the text descriptions of UI elements and the titles of application views to discover privacy menus hidden in apps.

In this work, we are first to tackle the web apps' scenario where dynamic content loading is a major challenge and where the UI views are not standardized. Moreover, we go beyond identification to the automated enforcement of web privacy controls.

Usability of privacy preferences: Previous research works have also aimed at understanding the usability of privacy preferences of online users. In particular, Ravichandran et al. [28] studied the burden associated with the availability of several privacy choices on social networking sites like Facebook and MySpace. In an empirical study, Habib et al. [11] studied a sample of 150 websites in which they assess the usability of the websites' data deletion options and opt-out for email communications and targeted advertising. In a followup work, Habib et al. [10] also conducted a field study to explore further the usability of these privacy choices from the perspective of end-users.

More recently, Kumar et al. [14] presented an integrated system to extract privacy choices from the privacy policies and present them to the user. They also conducted a field study of their extension. In this work, we neither restrict our analysis to opt-out pages nor assume that privacy control pages only appear in privacy policies. Furthermore, we go further in locating the fine-grained privacy options on the control pages and constructing control recipes that enforce the settings on-demand. By this, we tackle the last-mile problem in configuring privacy settings: how to go from the URL to discovering the options buried within the page.

10 Conclusion

In this paper, we present PriSEC, which automatically discovers, extracts, and presents the privacy settings for users. It also automatically enforces user preferences in a single interface. PriSEC uses machine learning techniques to create a machine-readable version of the privacy settings of any domain, thus enabling more efficient and usable user interfaces to be built. PriSEC overcomes the open nature of web development through novel algorithms that leverage the invariant behavior and rendering of webpages. We have evaluated the performance of PriSEC to find it accurately extracts and organizes the privacy controls of a given domain. Our user study showcases the usability improvement of PriSEC's interfaces.

Acknowledgement

We would like to thank the anonymous reviewers, Nina Taft, and Emily Stark for constructive comments on the earlier drafts of this paper. The work reported in this paper was supported in part by the NSF under grants 1838733, 1942014, and 2003129.

Availability

The datasets collected in this paper will be made available at https://github.com/wi-pi/prisec_data. We also plan to provide API access upon request for researchers to conduct further research utilizing the privacy settings recipes.

References

[1] Alessandro Acquisti and Ralph Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *International workshop on privacy enhancing technologies*, pages 36–58. Springer, 2006.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[4] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[5] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[6] Yi Chen, Mingming Zha, Nan Zhang, Dandan Xu, Qianqian Zhao, Xuan Feng, Kan Yuan, Fnu Suya, Yuan Tian, Kai Chen, et al. Demystifying hidden privacy settings in mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 570–586. IEEE, 2019.

[7] Hoa Trang Dang, Diane Kelly, and Jimmy J Lin. Overview of the trec 2007 question answering track. In *Trec*, volume 7, page 63, 2007.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[10] Hana Habib, Sarah Pearman, Jiamin Wang, Yixin Zou, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. "it's a scavenger hunt": Usability of websites' opt-out and data deletion choices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[11] Hana Habib, Yixin Zou, Aditi Jannu, Neha Sridhar, Chelse Swoopes, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. An empirical analysis of data deletion and opt-out choices on 150 websites. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019.

[12] H Harkous, K Fawaz, R Lebret, F Schaub, KG Shin, and K Aberer. Polisis: Automated analysis and presentation of privacy policies using deep learning. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018.

[13] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1681–1691, 2015.

[14] Vinayshekhar Bannihatti Kumar, Roger Iyengar, Namita Nisal, Yuanyuan Feng, Hana Habib, Peter Story, Sushain Cherivirala, Margaret Hagan, Lorrie Faith Cranor, Shomir Wilson, et al. Finding a choice in a haystack: Automatic extraction of opt-out statements from privacy policy text. In *The Web Conference (the Web Conf)*, 2020.

[15] Yee-Lin Lai and Kai-Lung Hui. Internet opt-in and opt-out: investigating the roles of frames, defaults and privacy concerns. In *Proceedings of the 2006 ACM SIGMIS CPR conference on computer personnel research: Forty four years of computer personnel research: achievements, challenges & the future*, pages 253–263. ACM, 2006.

[16] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[17] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*, pages 199–212, 2014.

[18] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. The privacy policy landscape after the gdpr. *Proceedings on Privacy Enhancing Technologies*, 2020(1):47–64, 2020.

[19] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*, pages 27–41, 2016.

[20] Bin Liu, Jialiu Lin, and Norman Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd international conference on World wide web*, pages 201–212, 2014.

[21] Yabing Liu, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing facebook privacy settings: user expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement*, pages 61–70. ACM, 2011.

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[23] Hector Llorens, Nathanael Chambers, Naushad UzZaman, Nasrin Mostafazadeh, James Allen, and James Pustejovsky. SemEval-2015 task 5: QA TempEval - evaluating temporal information understanding with question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 792–800, Denver, Colorado, June 2015. Association for Computational Linguistics.

[24] George R. Milne and Mary J. Culnan. Strategies for reducing online privacy risks: Why consumers read (or don't read) online privacy notices. *Journal of Interactive Marketing*, 18(3):15 – 29, 2004.

[25] Kaweh Djafari Naini, Ismail Sengor Altingovde, Ricardo Kawase, Eelco Herder, and Claudia Niederée. Analyzing and predicting privacy settings in the social web.

In *International Conference on User Modeling, Adaptation, and Personalization*, pages 104–117. Springer, 2015.

[26] Katarzyna Olejnik, Italo Dacosta, Joana Soares Machado, Kévin Huguenin, Mohammad Emtiyaz Khan, and Jean-Pierre Hubaux. Smarper: Context-aware and automatic runtime-permissions for mobile devices. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1058–1076. IEEE, 2017.

[27] Christian S Perone, Roberto Silveira, and Thomas S Paula. Evaluation of sentence embeddings in downstream and linguistic probing tasks. *arXiv preprint arXiv:1806.06259*, 2018.

[28] Ramprasad Ravichandran, Michael Benisch, Patrick Gage Kelley, and Norman M Sadeh. Capturing social networking privacy preferences. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2009.

[29] Joel R Reidenberg, N Cameron Russell, Alexander J Callen, Sophia Qasir, and Thomas B Norton. Privacy harms and the effectiveness of the notice and choice framework. *ISJLP*, 11:485, 2015.

[30] Joel R Reidenberg, N Cameron Russell, Vlad Herta, William Sierra-Rocafort, and Thomas Norton. Trustworthy privacy indicators: Grades, labels, certifications and dashboards. *Washington University Law Review*, 96(6), 2019.

[31] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[32] Kanthashree Mysore Sathyendra, Shomir Wilson, Florian Schaub, Sebastian Zimmeck, and Norman Sadeh. Identifying the provision of choices in privacy policy text. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2774–2779, 2017.

[33] Jeff Sauro. *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LLC, 2011.

[34] State of California. California Consumer Privacy Act (CCPA). https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375, June 2018. Assembly Bill No. 375.

[35] Christine Utz, Martin Degeling, Sascha Fahl, Florian Schaub, and Thorsten Holz. (un) informed consent: Studying gdpr consent notices in the field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 973–990, 2019.

[36] Ellen M Voorhees and L Buckland. Overview of the trec 2003 question answering track. In *TREC*, volume 2003, pages 54–68, 2003.

[37] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP-CoNLL*, volume 7, pages 22–32, 2007.

[38] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1077–1093. IEEE, 2017.

[39] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[40] Sebastian Zimmeck and Steven M Bellovin. Privee: An architecture for automatically analyzing web privacy policies. In *USENIX Security*, volume 14, 2014.

Appendix A Synthetic Data Set Details

We created a synthetic dataset to develop the visual classifier described in Sec. 5.2. We implemented *ReactJS* web application loaded with 11 popular React UI building libraries to generate the synthetic data. The list of all the libraries used is: *react-bootstrap, material-ui, semantic-ui, react-desktop, ant design, blueprintjs, shards-react, carbon-components-react, primereact, gestalt and grommet*. For each of the five UI component types, we traverse through each library's implementation and render a UI component for each available style the library offers.

Appendix B Details of the User Study

In this section we provide more details regarding the user study (Sec. 7.4) that we conducted to evaluate PriSEC.

We first ask the participants to fill the demographic information Then the participants' are asked to select the websites they are familiar with or have used in the recent times. The participants are required to select two websites and each participant is given two tasks for two websites that they are familiar with. Fig. **??** shows a snapshot of the question asked to the participant. We note here that if the participant select 'None of the above', then the participant exits the survey and is not considered a part of the study.

Figure 10: Snapshot of the question asking participants to choose familiar websites

Next, the participants are shown an example of the manual task and the plugin task on `bing.com`. The participants are then asked to complete the priavcy tasks. The snapshots for manual task and plugin task are shown in Fig. 11 and Fig. 12 respectively. We note a few key important things here:

- If the task is behind a login, the participants are asked to log in to the account before starting the task.
- Start of the task is considered after they navigate away from `https://example.com`
- The next button is only activated after the system verifies that the participant has completed the task.

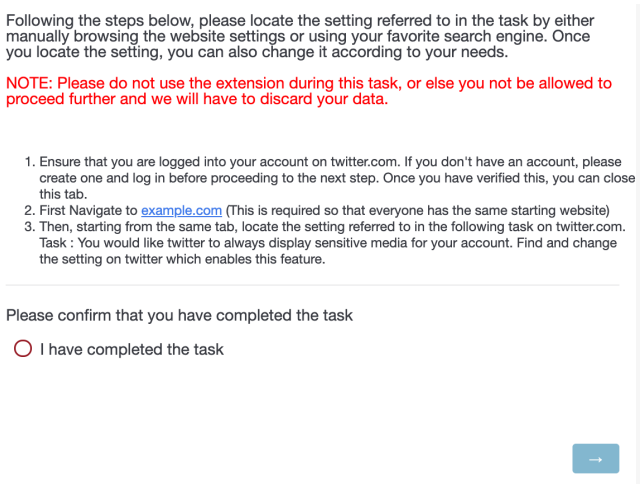


Figure 11: Snapshot of an example of manual task presented to the participant

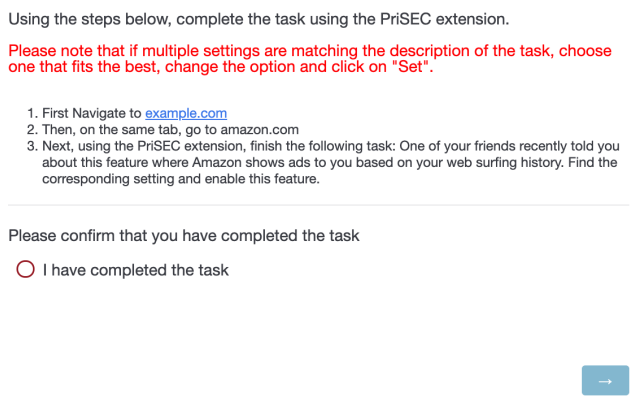


Figure 12: Snapshot of an example of plugin task presented to the participant

At the end of each task, the participants fill the System Usability Scale questionnaire [4]. A snapshot of the questionnaire for the manual task is shown in Fig. 13.

Finally, Fig. 14 shows a snapshot of the page which asks for the feedback about the study.

Appendix C Quotes from User study

Here, we present the list of answers that we received from the participants during the user study described in Sec. 7.4. These answers are in response to the open ended question shown in Fig. 14

1. The chrome extension seems to make it easier to find options that may be buried behind multiple clicks so it's pretty interesting in that regard.
2. This is a really neat feature! I just wish it would keep the tab open for me to review the setting change.
3. complex but interesting HIT

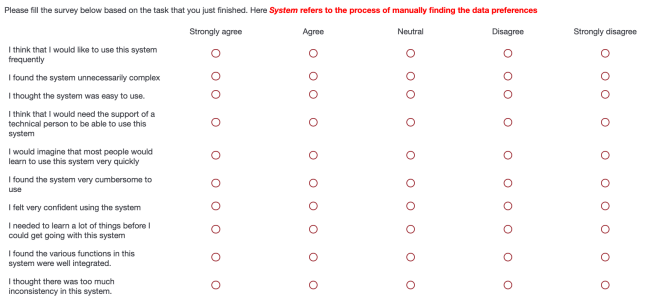


Figure 13: Snapshot of the SUS questionnaire for the manual task. The participants are asked to fill the questionnaire after each task.

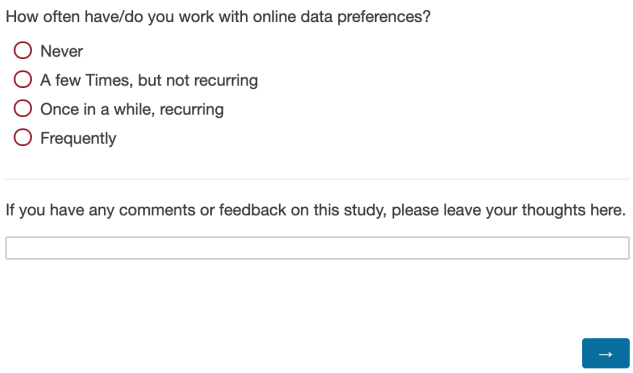


Figure 14: Snapshot of the feedback question presented to the participants at the end of the survey

4. I hope I did it right, but this extension was actually pretty cool. This is good for people who need to access these settings pretty quickly.
5. Amazing tool
6. Interesting study - thank you!
7. very nice task
8. It was fun and i enjoyed it
9. nice
10. THANKS FOR GIVING OPPORTUNITY TO TAKE PART IN YOUR STUDY
11. nice
12. All ok
13. i thought the extension was easy to use
14. I was surprised how tough it was to find that Twitter setting.
15. I was completing the qualtrics survey from firefox initially while doing task on chrome. On second task (twitter manual setting) survey said it didn't recognize my task, so I redid survey from chrome. I didn't change anything in survey.
16. interesting
17. was interesting thanks for the opportunity
18. None; I do love this extension though!
19. very nice task
20. Great survey!
21. I completed all tasks as instructed for this study. Thank you for this opportunity.
22. it was actually quite interesting thank you
23. It was good
24. Reddit parting is having some issue. I think the url parameters were not set correctly. After clicking the set button in plugin , it would redirect to "https://www.reddit.com/settings/privacy" which was no accessible. Error " Page not found". Rest everything went smooth.
25. A little complicated.
26. What a cool extension!!!! I love how I won't have to learn a new system every time I want to change a setting and I can just search the same way every time. Thank you!!
27. Thank you
28. This was interesting, thank you. Very nice plugin.
29. it awesome and it worth it
30. The example.com page would not load so I typed in the web address and accessed the site required on the tab that opened after I selected example.com. The Amazon task was difficult. There were only two options using PriSEC so I chose "show me interest based ads provided by Amazon". No results were given when I did a manual search for ads based on searches I have conducted on this browser. I also tried searching via Amazon settings but could not find a specific selection, but think I made the correct select using PriSEC, this task was a bit confusing.
31. Cool extension
32. interesting innovation
33. I like this product! It could definitely be useful.
34. cool beans
35. I am SO impressed!!
36. It was interesting
37. This extension is actually quite useful. I think I might just keep it on my google chrome.
38. Extension lead me to believe it was not working when setting Duck Duck Go HTTPs; when I clicked "set" it opened a new tab but quickly closed it. I looked in the extension and DDG HTTPS was disabled. Upon doing it a second time, it seems like it worked. This process could be improved upon a great deal.
39. Interesting service. No issues.
40. I turn off settings all of the time when signing up for new accounts.
41. I made an additional change to my Reddit settings as the first one I made was the wrong one. It was related but not exactly what was asked. I've never looked at those specific settings in Reddit before and found it difficult. I like the extension, though, and found it useful! Thanks for the HIT! I did my best!