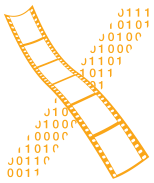

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Designing a Modular Browser Extension for Visual Countermeasures Against Dark Patterns

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Viola Valery Johanna Graf

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Ulrik Schroeder

Registration date: 08.01.2024
Submission date: 10.04.2024

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Contents

List of Figures

List of Tables

Abstract

Dark patterns are manipulative design patterns that aim to mislead users into actions that favour the service provider's interests, often at the user's expense. With the rise of such deceptive practices, active research is going into theoretical aspects like categorisation and underlying psychological aspects as well as practical solutions against dark patterns. This thesis deals with the practical side: applying visual countermeasures directly in the browser. Although there are existing browser extensions for this purpose, they are often times limited by using simple detection algorithms or countermeasures. In this project, a browser extension framework with the same purpose is developed, namely *Deceptive Defender*. It introduces a novel architecture design enabling flexible integration of two modules, one for implementing a detection algorithm and one for implementing an algorithm for creating countermeasures. This approach aims to address limitations of existing solutions, being adaptable to future developments of DPs, and to support future research on novel detection and countermeasure strategies.
We describe the development process of the framework, beginning with a requirement analysis, focusing mainly on inter-module and DOM communication. It then proceeds to present the architecture design and implementation of the framework's individual parts.
For evaluation, the framework is applied to a test webpage using hard-coded modules, implementing each three countermeasures against three dark pattern instances. This proof of concept validates the feasibility of the approach.
We conclude with a discussion and possibilities for future work, emphasising the need for the development of the corresponding modules and future enhancements of the framework itself.

Überblick

Dark Patterns sind manipulative Designmuster, die darauf abzielen, Nutzer zu Handlungen zu verleiten, die den Interessen des Dienstanbieters dienen, oft auf Kosten der Nutzer. Mit dem Aufkommen solcher Praktiken wird sowohl an theoretischen Aspekten wie der Kategorisierung und den zugrundeliegenden psychologischen Aspekten geforscht, als auch an praktischen Lösungen gegen Dark Patterns gearbeitet. Diese Thesis befasst sich mit der praktischen Seite: der Anwendung visueller Gegenmaßnahmen, die direkt im Browser angewandt werden. Obwohl es bereits Browser Erweiterungen zu diesem Zweck gibt, sind diese häufig durch unausgereifte Algorithmen zur Erkennung von Dark Patterns oder sehr simple und unerforschte Techniken für visuelle Gegenmaßnahmen eingeschränkt. In diesem Projekt wird ein Framework einer Browser Erweiterung mit demselben Ziel vorgestellt: das *Deceptive Defender* Framework. Es führt ein innovatives Architekturdesign ein, das eine flexible Integration von zwei Modulen ermöglicht; eines für die Implementierung eines Algorithmus zur Erkennung von Dark Patterns und eines für die Implementierung von visuellen Gegenmaßnahmen. Dieser Ansatz zielt darauf ab, die Einschränkungen bestehender Lösungen zu überwinden, anpassungsfähig an zukünftige Trends zu sein und die Forschung an neuen Strategien zur automatischen Erkennung und Gegenmaßnahmen zu unterstützen.
Wir beschreiben den Entwicklungsprozess des *Deceptive Defender* Frameworks, beginnend mit einer Anforderungsanalyse, die sich hauptsächlich auf die Kommunikation zwischen den Modulen und mit dem DOM konzentriert. Anschließend wird das Architekturdesign und die Implementierung der einzelnen Teile des Frameworks präsentiert. Zur Auswertung wird das Framework auf einer Testwebseite mit hartkodierten Modulen angewendet, die jeweils drei Gegenmaßnahmen gegen drei Instanzen von Dark Patterns implementieren. Dieser konzeptionelle Nachweis validiert die Machbarkeit des Ansatzes. Wir schließen mit einer Diskussion und schlussfolgern Möglichkeiten für künftige Arbeiten, welches notwendigerweise die Entwicklung der Module enthält, sowie die Weiterentwicklung des Frameworks selbst.

Acknowledgements

I would like to thank Prof. Dr. Jan Borchers and Prof. Dr.-Ing. Ulrik Schroeder for examining this thesis.

A special thanks to my supervisor René Schäfer, who sacrificed his time for my questions and discussions. You gave me the support and motivation I needed so that I felt confident throughout the process and also had a lot of fun.

I would also like to thank all people in the Media Computing Group for supporting me, especially Sarah Sahabi, Kevin Fiedler and Florian Plümäkers.

Finally, a big thank you to my (extended) family, my flatmates and to David, who have supported me over the last years and during this thesis project.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in British English.

Download links are set off in coloured boxes.

File: myFile[a]

[a] http://hci.rwth-aachen.de/public/folder/file_number.file

Chapter 1

Introduction

The internet has evolved from merely a few lines of text on a screen into a source of boundless information where visual and interactive elements dominate [Ross, 1995]. While companies are making use of it for online selling and advertisement, the web has grown into a crucial economic factor for many of them [Cockburn and Wilson, 1996]. Among these sophisticated user interfaces arises a trend towards the implementation of malicious designs on webpages [Lacey and Beattie, 2023, Utz et al., 2019], collectively referred to as *dark patterns*. These patterns aim at exploiting cognitive biases to manipulate user behaviour [Bösch et al., 2016, Waldman, 2020], often times benefitting companies at the user's expense [Mathur et al., 2019, Nouwens et al., 2020, Conti and Sobiesk, 2010].

DARK PATTERN / DECEPTIVE PATTERN (DP):
Harry Brignull was the first to define the term *dark pattern (DP)* on his website[1] in 2010. Later, the term was exchanged with the synonym *deceptive pattern (DP)* and is now defined as:

"tricks used in websites and apps that make you do things that you didn't mean to, like buying or signing up for something"[2].

Definition:
Dark Pattern / Deceptive Pattern (DP)

[1] darkpatterns.org *Published in 2010*
[2] deceptive.design (formerly darkpatterns.org) *Accessed: March 2024*

The use of DPs is widespread.

Studies by Lacey and Beattie [2023] and Utz et al. [2019] reveal a prevalence of DPs, with over half of the 100 most popular sites in New Zealand (54%) employing at least one, and approximately 57.4% of consent notifications in the European Union integrating one or more DPs.
Minor differences in text or design can already have a decisive impact on user interactions, for instance using technical language (”This site uses cookies” instead of ”This site collects your data”) [Utz et al., 2019]. Also commonly integrated in consent notifications are targeted differences in size and color of 'accept' and 'decline' buttons to influence user's decision process [Mathur et al., 2019, Utz et al., 2019]. Such a design instance would be categorised as *False Hierarchy* in Gray et al. [2018]'s taxonomy, or *Visual Interference*, following Mathur et al. [2019]'s taxonomy. Conti and Sobiesk [2010] argue based on their study with experienced users, that even when users are unconsciously aware of these patterns, they can have a psychological impact [Conti and Sobiesk, 2010, Waldman, 2020]. The informed user is likely to weigh up the costs and benefits of using websites that integrate manipulations [Conti and Sobiesk, 2010].

The HCI community is actively seeking lutions to inform and protect the user.

These developments are not only outraging consumers, referring to them as ”assholedesigns”[3], but also motivating the Human-Computer Interaction (HCI) community to actively seek solutions to inform and protect the user [Lukoff et al., 2021]. They have systematically approached this by gathering data about DPs [Mathur et al., 2019] and by categorising them [Gray et al., 2018, Chen et al., 2023, Mathur et al., 2021, Conti and Sobiesk, 2010]. To bring back the user's self-determination in the web, Mathur et al. [2021] propose the use of visual countermeasures and a possible design. Following this suggestion, Schäfer et al. [2023] created seven countermeasure designs and tested them in a user study.

Browser extensions are viewed as a promising tool to battle DPs.

This thesis deals with the approach of using visual countermeasures directly in the browser. Browser extensions (subsequently referred to as *extensions*) are often mentioned as a promising tool for this purpose [Conti and Sobiesk, 2010, Mathur et al., 2019, Bösch et al., 2016, Schäfer et al., 2023]. There are existing solutions, such as the Insite[4]

[3] Subreddit about DPs: 'r/assholedesign' *Accessed: March 2024*
[4] https://github.com/NicholasTung/dark-patterns-recognition *Ac-*

extension, developed at Teenhacks in Fall 2019 or the *Dapde Pattern Highlighter*[5] extension. Additionally, there are several extensions available in the Chrome Webstore[6]. It appears that existing extensions are either limited by only targeting cookie banners or certain websites, by their small number of covered DPs, or due to their naive detection and countermeasure techniques. For instance all of the extensions to our knowledge are using only one countermeasure style, which is highlighting the malicious elements and partly by additionally providing a popup when hovering over them.

Existing solutions have strong limitations.

This follows from the fact that research is still in early stages on detection algorithms and countermeasures [Gray et al., 2023a]. The fact new patterns continually emerge is not helpful, and it demands reactive detection- and countermeasure strategies [Hausner and Gertz, 2021].
It is inconvenient for the user to use each one extension for targeting Cookie Banners[7] and one to countermeasure the patterns *Bait and Switch* & *Hidden Information*[8].
To tackle named problems, this thesis is dedicated to the development of modular Chrome™ extension framework (namely *Deceptive Defender*), that enables flexible integration of a module for detection and a module for counter measuring. The detection module *Detector* can implement any detection algorithm that is based on the HTML and CSS data of the web page. The countermeasure module *Converter* can implement a wide range of countermeasure strategies. It uses the tools provided by the extension framework, such as inserting/deleting elements or modifying attributes and styles of the websites elements.

Deceptive Defender framework aims at enabling flexible integration of detection and countermeasure strategies.

The *Deceptive Defender* framework aims at simplifying testing and updating of detection algorithms and countermeasures. It assures adaptability to future trends and scientific developments with its modular structure. For evaluation, selected visual countermeasures will be implemented exemplarily on a webpage.

---

cessed: March 2024

[5] https://github.com/Dapde/Pattern-Highlighter *Accessed: March 2024*

[6] https://chromewebstore.google.com *Accessed: March 2024*

[7] https://github.com/wsg-ariadne/ariadne *Accessed: March 2024*

[8] https://github.com/Carmineh/Dark-Pattern-Identifier *Accessed: March 2024*

Chapter 2

Related Work

2.1 Taxonomies & Prevalence

The UX designer Harry Brignull was not only the first to use the term DP, but he set the stage for public awareness and academic inquiry into DPs. He wanted to educate the public by launching a website[1], providing both a list of DPs with explanations and a 'hall of shame' showing DP examples and shaming companies for the use of malicious designs. One example is illustrated in Figure 2.1. Raising awareness is an effective way of battling DPs [Conti and Sobiesk, 2010, Bongard-Blanchy et al., 2021]. When awareness rises, the user tends to view integrations of malicious designs more negatively [Commission et al., 2022]. However, they can still psychologically impact users, even when the user is subconsciously aware of them [Conti and Sobiesk, 2010, Waldman, 2020]. In such cases, informed users may begin to carefully consider the trade-offs of interacting with websites that employ these manipulative tactics [Conti and Sobiesk, 2010].

Later, researchers followed the initial step taken by Harry Brignull and created taxonomies [Conti and Sobiesk, 2010, Gray et al., 2018, Mathur et al., 2019, Chen et al., 2023] to

The initial step is to create taxonomies.

[1] https://www.deceptive.design/ *Accessed: March 2024*

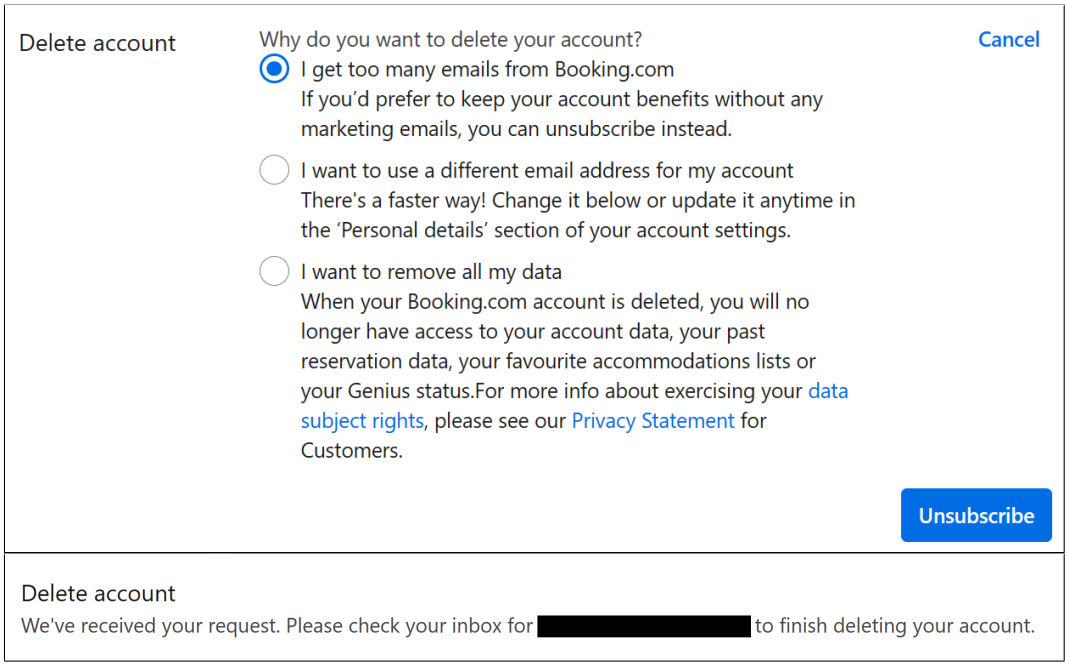

Figure 2.1: Dark pattern *Immortal Accounts* on booking.com[a] as described by Bösch et al. [2016]. After clicking on the button *delete account*, the user is presented with three additional options. The user is forced to read the options before realising that only the third choice actually deletes their user account. After selecting the correct option, the service provider continues to increase the workload for the user by forcing a further confirmation via email.

[a] https://www.booking.com/ *Accessed: March 2024*

grasp the scope of application and variety of DPs.

But these practices are not only used on websites. Conti and Sobiesk [2010] formulated a categorisation based on a long-term study that involved websites, desktop software and other media. For instance, designing uninstalling of an application difficult is categorised as a DP by Conti and Sobiesk [2010] and is a specialty of mobile applications and can be encountered when trying to remove an operating system's default application. More recent, Chen et al. [2023] published a research paper that was solely based on mobile applications, including a taxonomy.
In a widely popular work by Gray et al. [2018], two researchers collected 118 artefacts from selected online

platforms. While trying to categorise them, they found that some content needed either more specific or broader categories than the ones Brignull had published on his website. Therefore, they expanded Brignull's concept and named 5 broad categories of DPs: *Nagging, Obstruction, Sneaking, Interface Interference* and *Forced Action*, with more specific sub-categories.
Mathur et al. [2019] used a crawler on 11,000 websites and collected 1,818 instances of DPs on approximately 11% of the websites. When targeting specifically shopping websites, it is likely to encounter an instance of *Interface Interference* or *Forced Action* [Lacey and Beattie, 2023].

A consensus on the terminology of DPs is needed.

There is a great deal of variation in the creation of taxonomies and sometimes two authors will choose a different name for the exactly the same category. For example, Conti and Sobiesk [2010] name categories such as *Forced Work* and *Interruption*, which correspond to *Nagging* and *Obstruction* in Brignull's taxonomy[2]. Existing taxonomies are also used as a basis and extended or refined [Chen et al., 2023, Gray et al., 2018, Mathur et al., 2019], or a different target is addressed (e.g. mobile applications) [Conti and Sobiesk, 2010, Chen et al., 2023]. This calls for a consensus about the categories of DPs. Gray et al. [2023b] have already published a preliminary three-level ontology aimed at supporting translational research and regulatory action. Soe et al. [2022] mention that such a codebook consensus would need to "account[..] for newly identified patterns in future work"[Soe et al., 2022].

Lacey and Beattie [2023] selected the top 100 New Zealand websites using Alexa rankings to investigate the occurrence and clustering of DPs. More than half of these websites (54%) implemented at least one DP [Lacey and Beattie, 2023]. Based on a screening of 6,146 cookie banners, Coudert [2020] found instances of *Obstruction* or *Interface Interference* [Gray et al., 2018] on 89.63% of them.
Lacey and Beattie [2023] pinpointed that DPs are frequent during purchases, on homepages and when cancelling services. It is likely to encounter fake timers, pressuring

[2]https://www.deceptive.design/ *Accessed: March 2024*

the user during purchases, and forced registrations on e-commerce sites. Mathur et al. [2019] state that the most common types of DPs are covert, deceptive, or hiding information. Many are aimed at influencing the user, using cognitive biases [Gray et al., 2018, Bösch et al., 2016, Waldman, 2020].

DPs exploit cognitive biases.

COGNITIVE BIASES:
Kahneman [2003] describes cognitive biases as systematic patterns of deviation from rationality in judgment. Biases stem from the mind's attempt to simplify information processing through heuristic thinking, due to insufficient capacity for comprehensive analysis before making decisions.

Definition: *Cognitive Biases*

Partly, these psychological effects are described when creating taxonomies [Bösch et al., 2016, Mathur et al., 2019]. According to Waldman [2020], the five most persuasive biases are anchoring, framing, hyperbolic discounting, overchoice, and metacognitive processes in decision-making. Another example is the state of cognitive dissonance, which is likely to emerge when the user fails to delete their account due to the service provider designing the process of deletion to be challenging [Bösch et al., 2016].

COGNITIVE DISSONANCE:
According to Festinger [1957], cognitive dissonance occurs when an individual experiences a conflict between two or more cognitions (thoughts, beliefs, or attitudes), where one cognition directly contradicts another. This state of psychological tension is uncomfortable for the individual and motivates them to reduce it by changing their cognitions, justifying their behaviour [Festinger, 1957].

Definition: *Cognitive Dissonance*

This state of psychological tension may cause the user to reconsider their original intention to delete the account, in order to reduce the inconsistent mental state they are in (see Figure 2.1).

2.2 Automated Detection

The taxonomies discussed in Section 2.1 are partly based on (semi-)automated detection of DP instances [Mathur et al., 2019, Lacey and Beattie, 2023]. Automatic detection not only facilitates analysing the prevalence of DP and categorising of DPs, but can be used as a tool to enable the application of countermeasures directly in the browser.

Text-based detection of DPs is promising.

Two fundamentally different approaches can be used for data retrieval: analysing the Document Object Model (DOM) of a website and using visual detection. Detecting DP instances in screenshots has the advantage of generalisability across platforms, while still being able to extract textual information [Chen et al., 2023]. Considering today's websites are heterogeneous, detecting text-based DPs is promising since text can be identified on almost any website [Hausner and Gertz, 2021].

Mathur et al. [2019] found DP instances on approximately 11% of the analysed websites, by solely analysing text contents. They emphasise, that this number constitutes a lower bound due to their restricted detection method. The detection of DPs using solely textual contents introduces difficulties in maintaining low false positive rates, due to the context-dependent nature of most DPs [Hausner and Gertz, 2021].

As a part of a master's thesis project, Coudert [2020] collects a dataset of cookie banners, trying to adapt the methodology of Mathur et al. [2019]. The authors uses a segmentation algorithm derived from Mathur et al. [2019], and then assigns a score to each segment, based on "typical" vocabulary used in cookie banners [Coudert, 2020].

Soe et al. [2022] trained a prediction model on 300 websites to detect DPs on cookie banners. Before being fed into the prediction model, the data needs to be processed into a set of feature values.

Prediction models are fed with feature values of DOM elements.

Recently, Chen et al. [2023] identified six key characteristics for such a prediction: coordinates and element types, text content, status (e.g., of checkboxes), icon semantics, text colours, and relationships between UI elements. They developed a detection tool specifically for mobile applications, and propose using it for a one-time screening to

determine if the user wants to make use of a particular app [Chen et al., 2023].

Some DPs may be undetectable.

Soe et al. [2022] note that the complexity of information presented to users makes it is difficult for both humans and machines to read the intent behind it [Soe et al., 2022]. Curley et al. [2021] state that some DPs are even undetectable. This may be due to the variety in which they occur [Curley et al., 2021], and the rarity of some cases [Hausner and Gertz, 2021]. Hausner and Gertz [2021] propose exploring rare pattern detection and unbalanced classification solutions. They themselves chose graph neural networks for their detection, leveraging the natural tree structure of the DOM for more effective pattern recognition. Another problem are the high false positive rates of existing approaches Coudert [2020].
It should be noted, that most of the detection approaches are semi-automatic and only used for dataset creation [Coudert, 2020, Soe et al., 2022]. Most would not suffice as a foundation to visually countermeasure DPs in the web. Gray et al. [2023a] report that work on deployment of automated detection techniques is generally not common.

2.3 Countermeasures

Bongard-Blanchy et al. [2021] propose different intervention measures against DPs; educational-, design-, technical- and regulatory measures. Additionally, the authors discuss four different aims of these measures: DP Awareness, DP Detection, DP Resisting and DP Elimination. An educational measure within the scope of DP Awareness would be for instance the education of developers to use more user-friendly designs [Gray et al., 2018, Lukoff et al., 2021]. However, this approach may not be effective when these designs are used intentionally. Design patterns that have a negative impact, whether intentional or not, are commonly referred to as *Anti-Patterns* [Greenberg et al., 2014]. The intervention measure can be aimed at educating users too. Bongard-Blanchy et al. [2021] claim, that when a user recognises a DP instance, they are less likely to be influenced by it.

Early on, Conti and Sobiesk [2010] conducted a survey, asking 47 security experts to rate the ease of use and effectiveness of seven available counter measuring tools such as pop-up blockers, which is a technical intervention measure for the elimination of DPs within the dimensions of Bongard-Blanchy et al. [2021]. The browser-plugin was identified as the most effective at that time, but the authors emphasise the need for more effective ways of counter measuring.

Varied countermeasuring tools were conducted early-on.

Later, Bösch et al. [2016] delved into privacy-related DPs, recommending tools for specific privacy concerns. These correspond partly to the category of technical intervention measures by Bongard-Blanchy et al. [2021], with the scope DP Resistance. One example is the *Privacy Bird* software, which is a tool that helps the user identify, whether the a website is conform with the user's security preferences.

As mentioned in Section 2.1, Mathur et al. [2019] propose leveraging their data set to develop tools to counteract DPs, specifically mentioning browser extensions (subsequently referred to as *extensions*) to detect and visually countermeasure them [Mathur et al., 2019]. Recently, Schäfer et al. [2023] expanded on the proposed design concept of Mathur et al. [2019].The authors conducted a user study on visual countermeasures, exploring six different designs applied to three DPs: *Low-stock Messages*, *Visual Interference*, and *Confirmshaming*. The most promising countermeasure design from a user perspective was highlighting the respective elements plus providing an explanation using a popup (also known as Highlight with Explanation (HL+E)) shown in Figure 2.2 [Schäfer et al., 2023]. Completely hiding the DP (also known as Hide without Marking (HD)) without the user's knowledge was seen as controversial, due to lack of transparency and distrust of the user. According to Schäfer et al. [2023], some countermeasures might be more useful for specific application areas, such as the countermeasure Lowlight (LL) (making the malicious content less noticeable) being helpful against *Low-Stock Messages* and HL+E or HD being better for *Confirmshaming* and *Visual Interference*.

There is already a study on visual countermeasure designs.

Corresponding to the categorisation by Bongard-Blanchy et al. [2021], browser extensions are a technical intervention method. The authors assigned this measure to the scope of DP Elimination. But there is the possibility for browser

Browser extensions to raise awarness about and eliminate DPs are available.

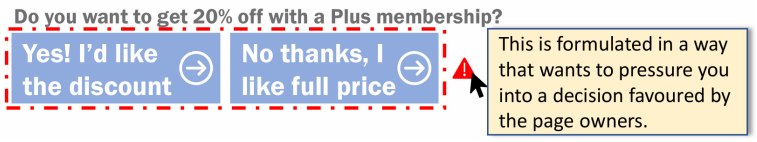

Figure 2.2: Countermeasure Highlight with Explanation, tested in a user study by Schäfer et al. [2023].

extensions just to aim at raising awareness (intervention scope DP Awareness), taking browser extensions such as *My cognitive bias*[3] or *Brainy Tab*[4] as a role model. These extensions aim to educate the user about cognitive biases that underlie many DPs (as outlined in Section 2.1).

The use of browser extensions with the aim of eliminating DPs (intervention scope DP Elimination [Bongard-Blanchy et al., 2021]) was mentioned by Mathur et al. [2019] as a tool to apply countermeasures directly in the web browser. There are existing projects for this specific purpose, published on GitHub[5] (under the keyword "dark+patterns") and the Chrome™ Webstore[6].
Some of them focus on one or two DP types, such as the *Dark Pattern Identifier*[7] and the extension *Trick Question Detection*[8], both developed by students of the University of Salerno. Another extension is even limited to one website[9], specifically helping the user to identify sponsored products, which is designed to be difficult (categorised as the DP *Disguised Ad* by Gray et al. [2018]).
Ariadne[10] is a browser extension, developed as a part of a research project by students of the University of Philippines - Dilimanon. This extension is available in the

[3] https://chromewebstore.google.com/detail/my-cognitive-bias/cmapeoagadpppgajnicpagcgpdklfhch *Accessed: March 2024*
[4] https://microsoftedge.microsoft.com/addons/detail/brainytab/hepiifekbekhoabjbnapgpdjdgkmhgpf *Accessed: March 2024*
[5] https://github.com/ *Accessed: March 2024*
[6] https://chromewebstore.google.com/ *Accessed: March 2024*
[7] https://github.com/Carmineh/Dark-Pattern-Identifier *Accessed: March 2024*
[8] https://github.com/xrenegade100/trick-question-detection *Accessed: March 2024*
[9] https://github.com/keybraker/reSkroutzed *Accessed: March 2024*
[10] https://github.com/wsg-ariadne/ *Accessed: March 2024*

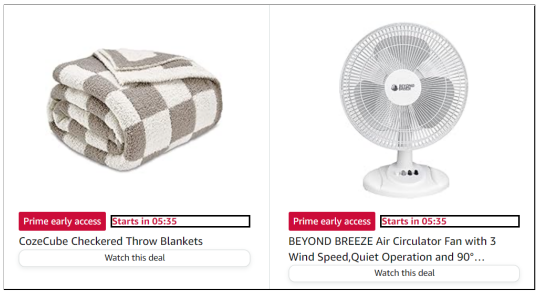

Figure 2.3: Example of a countermeasure, applied by the *Dapde Pattern Highlighter*[a] browser extension on amazon.com[b], outlining two counters in black.

[a]https://github.com/Dapde/Pattern-Highlighter *Accessed: March 2024*

[b]https://www.amazon.com/ *Accessed: March 2024*

Chrome™ Webstore and is aimed at detecting deceptive designs in cookie banners such as *Unclear Language*, using a Naive-Bayes classifier, and *Weighted Options*, using an image classifier.
The extension *Insite*[11] was developed by the winning team of Teenhacks[12] in 2019, building upon the segmentation algorithm and dataset of Mathur et al. [2019] for automatic detection of DPs. *Insite* highlights identified DP instances and provides popups explaining the category of the detected DP. The extension relies on a back-end server for the use of their Bernoulli Naive-Bayes model for text classification. This server must be launched by the user in order to enable detection. It theoretically covers all DP categories present in Mathur et al. [2019]'s dataset: *Sneaking, Urgency, Misdirection, Social Proof, Scarcity, Obstruction* and *Forced Action* [Mathur et al., 2019].

As a part of the Dark Pattern Detection Project (DAPDE)[13], a collaborative project between the Institute of Computer Science at the University of Heidelberg and the German Research Institute for Public Administration, a similar extension was developed: The *Dapde Pattern Highlighter*.

[11]https://github.com/NicholasTung/dark-patterns-recognition *Accessed: March 2024*

[12]https://teenhacksli.com/ *Accessed: March 2024*

[13]https://dapde.de/de/ *Accessed: March 2024*

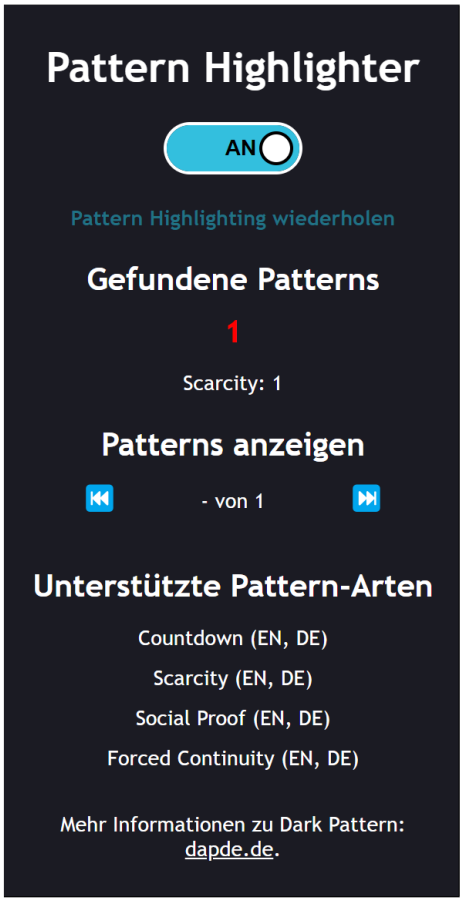

Figure 2.4: Popup of the *Dapde Pattern Highlighter*[a] browser extension, including a button to toggle detection on/off and a counter of dark pattern instances found on the respective webpage.

[a] https://github.com/Dapde/Pattern-Highlighter *Accessed: March 2024*

This extension implements graph neural networks on the Document Object Model (DOM), using regular expressions for detection. To allow the detection of a countdown on e.g. shopping websites, two temporary copies of the webpage are created and analysed (see Figure 2.3). The extension also shows a counter for the number of DP instances found in one tab (see Figure 2.4).

Chapter 3

Introducing the Browser Extension Framework *Deceptive Defender*

Reflecting on the existing body of work, there is an increasing interest to utilise browser extensions to deploy visual countermeasures against DPs. In the following, the advantages of designing a modular browser extension for the HCI research community are outlined.
Unlike existing solutions, we aim for the design of a modular browser extension by implementing an extension framework, which features two interfaces for a separate detection module (*Detector*) and a countermeasure module (*Converter*). These modules are not part of this work and will be hard coded for testing purposes. The modular architecture of *Deceptive Defender* offers several advantages for future research:

The modular design offers advantages for future research.

1. **Flexibility in testing:** By decoupling the detection and countermeasure processes, independent testing of new approaches for both processes is possible.

2. **Future-proofing:** The possibility to adapt to future developments easily, whether to expand the set of detectable DPs or to introduce new countermeasures.

Considering the current state of research in detection algorithms and visual countermeasures (outlined in Section 2), the *Deceptive Defender* aims to establish a basis for the HCI community to trial novel methods. Existing extensions are often restricted to specific types of DPs, targeting only one website, or relying on naive detection techniques. Typically, these extensions employ the same countermeasure: highlighting malicious elements.
The framework enables other developers to plug in their custom *Detector* and *Converter* modules, facilitating experimentation and the deployment of novel solutions without the need to build a new browser extension from the ground up. Utilising the independence of the modules, interactive prototypes can be used for in-depth evaluation of countermeasures in user studies, which was proposed by Schäfer et al. [2023] as the next step.

3.1 Requirement Analysis

In the following, we will outline the functional and non-functional requirements for a successful implementation of our modular concept. We do not follow a specific methodology, doing an in-depth analysis of every part of the implementation, but rather concentrate on the main actions and features that are required to be implemented by the framework. Also we are elaborating on the data requirements of the modules, as well as functionalities that have to be provided at their interfaces to ensure a seamless integration of the *Detector* and *Converter* module.

Functional requirements:

1. User interaction
    - Activate and deactivate detection
2. Communication with the Document Object Model
    - Extracting HTML content
    - Extracting CSS content
    - Injecting stylesheet
    - Adding/deleting elements
    - Changing attributes and styles
3. Communication between scipts/modules
    - Assigning data to the correct tab
    - Enabling referencing each element individually
    - Extensibility

Non-functional requirements:

1. Security
2. Documentation

The user should be able to detection on and off.

The initial functional requirement is to enable **user interaction**. The user should be able to control the detection process by toggling it on or off based on individual preferences. This state needs to be saved across browser sessions. Keeping it simple is due to the limited time frame, with priority given to the following requirements.

HTML and CSS data is essential for informed decision-making.

The **communication with the Document Object Model** (DOM) includes reading the HTML and CSS content, which is essential for the *Detector* to make informed decisions. With this information, the *Detector* can consider element styles and attributes as well as placement and relationships between elements. Both HTML and CSS content can be used to determine visual characteristics of

elements. Thus, both files are crucial for the detection of DPs based on visual cues, and possibly for the *Converter* when trying to match the design of two related elements.
The *Deceptive Defender* framework is required to handle the injection of changes that are requested by the *Converter*. The functionalities to be provided at the *Converter* interface should cover all necessary actions to implement a large set of visual countermeasures. This framework aims specifically at being able to implement the countermeasures created and tested by Schäfer et al. [2023]. The required functions to be implemented by the framework for each of the countermeasures by Schäfer et al. [2023] are listed in Table 3.1.

A wide range of possible content alterations is required.

Countermeasure	Required function
Highlight with Explanation (HL+E)	ADD
Highlight without Explanation (HL)	ADD
Lowlight (LL)	ADD, STYLE, ATTR
Hide without Marking (HD)	DEL, STYLE, ATTR
Hide with Marking (HD+M)	DEL, ADD, STYLE, ATTR
Switch (SW)	DEL, ADD, STYLE, ATTR

Table 3.1: Countermeasures designed by Schäfer et al. [2023] and their required functions for injection, listed using the following abbreviations: *ADD* for adding a custom element to the webpage, *DEL* for deleting an element from the webpage, *STYLE* for changing style property of one or more elements and *ATTR* for changing attribute property of one or more elements.

The requirements in Table 3.1 are simplified and depend on the implementation of the *Converter*.
The countermeasures Highlight with Explanation (HL+E), Highlight without Explanation (HL), Hide with Marking (HD+M) and Switch (SW) require adding elements, such as a container for enclosing two elements that should be highlighted together (see Figure 2.2), adding icons for indicating an available popup or buttons to implement the switch functionality for the countermeasure SW proposed by Schäfer et al. [2023]. The countermeasure LL requires at least ADD, STYLE and ATTR, for instance in case of a button label text being implemented as an *innertext* attribute. For instance, if a button label text is implemented

as an *innertext* attribute, the text has to be featured through two or more additionally added elements, which allow changing STYLES (e.g. text transparency levels or colours) separately for two or more parts of the text content.
The countermeasures HD, HD+M and SW require either the action of deleting (DEL) **or** changing styles and attributes of elements, depending on whether the attribute visibility is changed to *hidden* (and back to *visible* in case of SW), or the element is deleted completely (and then added again in case of SW).
Additionally, there may arise the necessity to utilise the options STYLE and ATTR for the implementation of HL and HL+E too, in order to correct the placement after adding or deleting other elements.
To sum up, it should be possible to execute all the countermeasures proposed by Schäfer et al. [2023] with the functions DEL, ADD, STYLE & ATTR. Hence, these functions are mandatory to be incorporated in the framework. Further expansion of these functionalities is discussed in Section 6.

There are special requirements for the process of adding elements.

The function ADD underlies special requirements for implementation. Firstly, the framework must be able to provide a proper selection of element types to be creatable, for example div-elements or p-elements. Furthermore the *Converter* module must be provided with placement options of these created elements as parents or child elements of already existing ones. When placed as a child element, it should be possible to position it in a precise ordinal rank among other child elements.

The countermeasures HL+E, LL and SW are used for the evaluation.

For evaluation, the framework should be evaluated using hard-coded modules, applying the countermeasures HL+E, LL and SW on selected DPs. This selection of countermeasures is justified by the fact that the remaining ones by Schäfer et al. [2023] that we discussed are merely variations of these three and do not offer any additional value for our proof of concept. HL is covered by HL+E and SW implements HD with the additional function to reverse back to the original state. Also the principle of marking the area where the countermeasure HD was applied (namely HD+M) is basically the countermeasure HD plus highlighting the area as in HL.

Another essential requirement is a smooth **communication** between the modules and the *Deceptive Defender* framework. The framework has to provide clear interfaces and minimise the requirements for module implementation.
In theory, there must be at least each two data streams (back and forth) between the modules and the service worker. First, there is the outgoing data to the *Detector*. As already outlined, this module requires at least the HTML and CSS contents. Additionally, because the framework must be able to handle requests from multiple tabs, there has to be some kind of identification inside of the payload of each message, to identify, which tab the message belongs to.
Next, there is the data stream back from the *Detector* to the service worker. This detection data is essential for the *Converter*. However, the modules should not communicate directly with one another, to ensure that the framework's interfaces and the requirements for the modules are untouched by changes in the modules. To outline the requirements for the format for (indirect) communication between the modules is challenging, because it depends very much on the detail with which the detection and the countermeasure process is implemented. For the purpose of this proof of concept, the requirements are kept low: The *Detector* must be able to point out detected elements individually and to name the category of DP they belong to. Additionally to the outgoing data of the *Detector*, the *Converter* requires the HTML and CSS contents to consider the original state and context of the detected elements while choosing appropriate alterations.
Lastly, the outgoing data of the *Converter* must contain detailed instructions on how to implement changes on the website that are needed for effective countermeasures. The format of the outgoing data of the *Converter* is most sensitive to mis-planning because of possible dependencies between different element inside of an enquiry. For instance when this module commands the creation of an element, it has to be considered that it can be object of style changes or must serve as the parent element of other to be added elements in the same enquiry.

The communication format must be extensible.

In general, the definition of data formats for this communication harbours challenges. The data has to be clearly arranged and the process of encoding and decoding easily

comprehensible. The chosen format for communication should be extensible, considering that this thesis project serves merely as a design outline of a modular browser extension and a proof of concept. It is possible that not all requirements will be met, creating the necessity to extend the data formats because of unpredictable challenges when implementing the *Detector* and *Converter*.

On the non-functional front, **security** is factor that must be considered. The biggest concern would be injection of malicious code. Counteractions in the implementation can be for instance minimising the manifest permissions or restricting capabilities of the modules to communicate with the DOM directly.

Code Documentation, the second non-functional requirement, is important for the extension's further development. This includes both commenting the codebase and providing information that is helpful for the implementation of compatible *Detector* and *Converter* modules. Also the framework itself probably can be enhanced afterwards and therefore should be comprehensible.

3.2 Architecture Design

This chapter outlines the conceptual design and architecture of the Chrome™ extension framework *Deceptive Defender*, discussing specific roles and contributions of the framework itself and the connected *Detector* and *Converter* (in the following referred to as modules).

Figure 3.1 illustrates the data flow between the following components: DOM, extension framework, *Detector* and *Converter*. The core part, labelled as *Deceptive Defender Framework* in Figure 3.1, consists of the popup, the service worker and a content script. The framework acts as the central command, fulfilling the requirements set in Section 3.1:

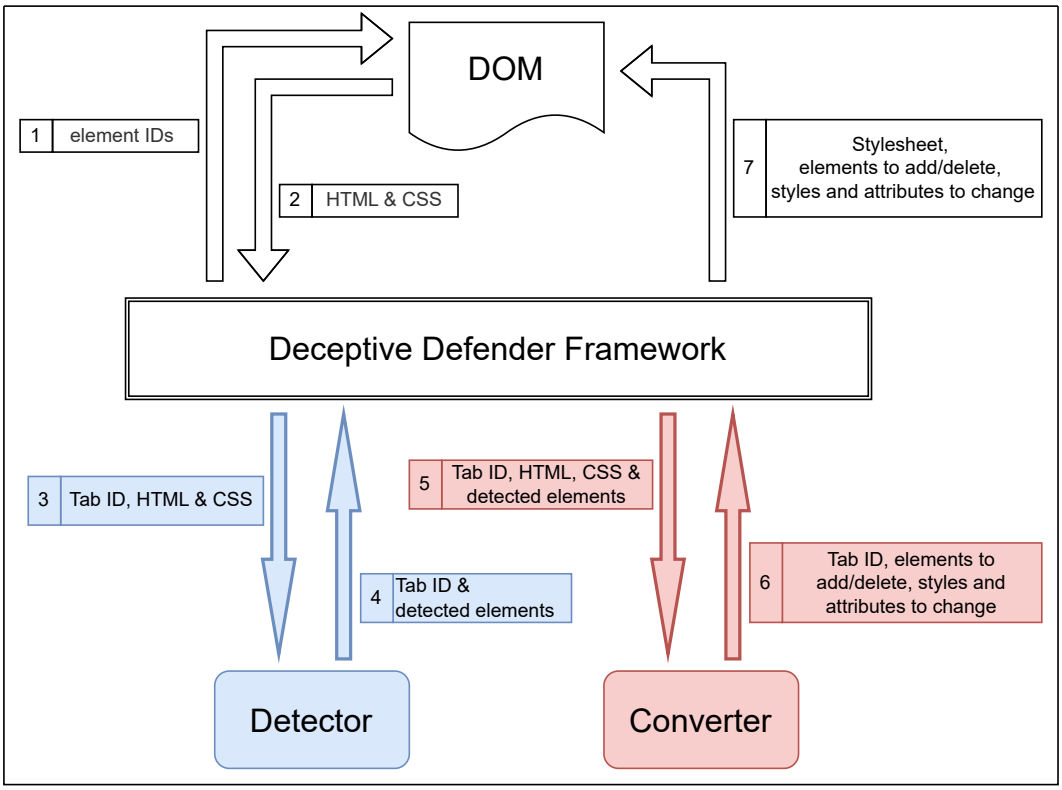

Figure 3.1: Diagram showing the data transfer between: DOM, Extension Framework, Detector module and Converter module.

1. User interaction
2. Communication with the Document Object Model
3. Communication between scripts/modules

The responsibility of the popup is the deactivation and activation of the extension's feature. The implementation of the popup will be explained in detail in Section 4.3. One run-through for the application of countermeasures on a specific tab starts for example when the popup signalises it.

The content script handles any communication with the DOM, except the injection of the stylesheet.

The only content script used in this framework (referred to as *the* content script hereafter) arranges the injection of unique identifiers for each element on the webpage for the identification of specific elements during detection

or conversion processes (labelled as the second step in Figure 3.1). Furthermore, as the second step, it extracts the necessary data for detection, enabling a thorough analysis of the webpage's structure and styling by the *Detector*.

The framework uses Chrome's™ Manifest Version 3.

The extension is build on Chrome's™ Manifest Version 3 (MV3)[1], the latest evolution in the framework governing Chrome™ extensions. In accordance with this version, the extension framework employs a service worker to handle various background tasks, essential for its operation (see Section 4.4). This includes managing requests by multiple tabs in parallel and ensuring that the data is routed to the correct instance of a content script.

The *Detector* identifies DP instances.

Subsequently, in the third step, the service worker sends the required data to the *Detector*. The *Detector*'s primary function is to analyse the information provided by the extension framework for elements that represent a DP (see Section 4.6.1). Upon detection, it communicates the identified deceptive elements back to the extension's service worker in step 4, and the services of the *Converter* are demanded by the service worker in step 5 (as illustrated in Figure 3.1).

The *Converter* decides how to counter act visually.

Receiving the detected elements, the *Converter* is tasked with deciding how to counteract (see Section 4.6.2). It determines which elements need style or attribute modifications and whether elements should be added or deleted. The *Converter*'s output includes instructions on these changes, which are then sent back to the framework. The injection of a custom stylesheet is done by the service worker of the framework, and the implementation of these changes in the DOM is handled by the same content script as before (see step 7 in Figure 3.1).

The JSON format is used for inter-module communication.

The choice of our data transfer format between the extension framework and the modules is the JSON format. This communication embodies the requirements for modularity and extensibility (see Section 3.1), being adaptable to potential detection techniques, future trends of DPs and countermeasures.

[1] https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3 *Accessed: March 2024*

The *Detector* and *Converter* modules are both hard coded in the course of this thesis, as they are intended to be exchangeable by other developers through their respective approaches. Further explanations regarding their integration into the framework and the required data formats will be worked out in Chapter 4.

Chapter 4

Implementation

This chapter discusses the practical realisation of the *Deceptive Defender framework*, translating the conceptual designs from Section 3.2 into working components, while addressing the requirements, analysed previously in Section 3.1.
The structure of the following sections largely mirrors the sequence of actions outlined in Figure 4.1. Central to our discussion are the roles of the core components and their communication: the DOM, the extension framework, and both modules (illustrated in Figure 3.1).

4.1 Security

The factor **security** was considered during development, among others, by following a guide for security in browser extensions by Chrome™[1].
The extension is built on Chrome's Manifest Version 3 (MV3), which introduces significant improvements in security, privacy, and performance compared to its predecessor. MV3 enhances security through more granular permission controls and the introduction of service workers for

[1]https://developer.chrome.com/docs/extensions/develop/security-privacy/stay-secure *Accessed: March 2024*

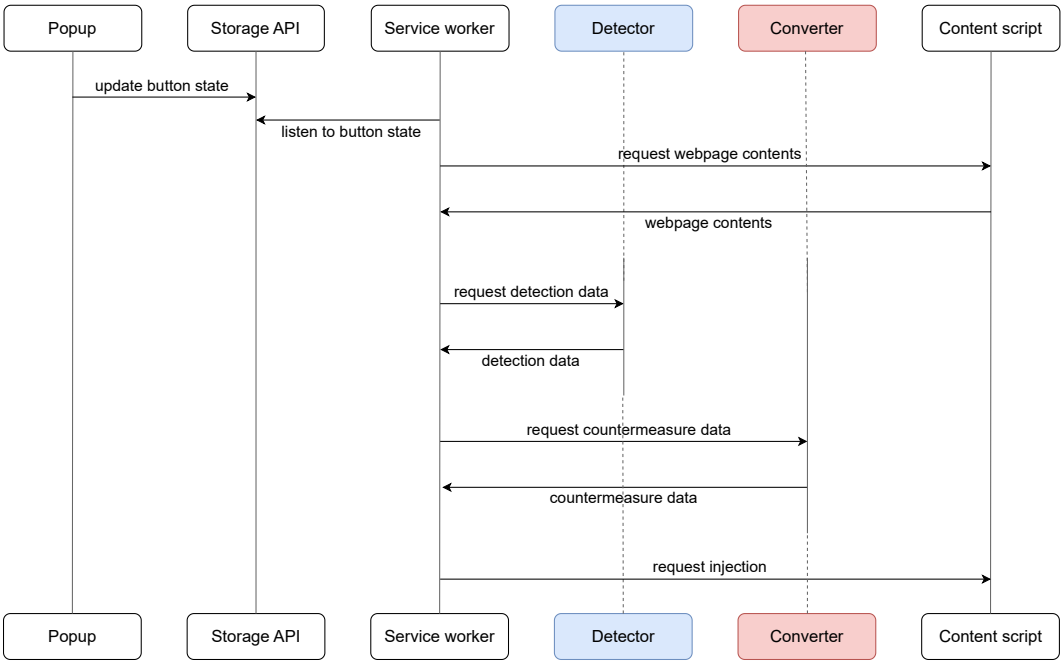

Figure 4.1: Action diagram showing the sequence of actions for one run-trough. Including actions between the popup scripts, the storage API (*chrome.storage*), the service worker, the content script, the *Detector* and the *Converter*.

background processes, replacing the previously used background pages. This change mitigates the risk of excessive resource usage and potential privacy infringements by ensuring better controlled background activities.

The *Deceptive Defender* framework requires minimal permissions.

The manifest permissions used by our framework are minimal. With that, the intensity of attacks can be reduced because every attacker is bound by the same rules [Aravind and Sethumadhavan, 2014].
The permission *activetab* is used because unlike broad permissions that grant access to all the websites a user visits, it provides access only to the current tab. The permission *activetab* still ensures that the extension can perform its intended functions while minimising unnecessary access to user data. In Section 4.3, the purpose of the permission *tabs* is explained in context. It is used, for example, to to reload a webpage. The third permission is *scripting*, used to wake up scripts before sending requests, to make sure they are processed accordingly. The last permission is *storage*,

facilitating the usage of the (*chrome.storage*) API[2].

4.2 Manifest

The manifest file is a fundamental component of this extension framework. It defines the service worker and content scripts and configures resources and the previously outlined permissions. The manifest is in accordance with MV3.
If necessary, the host permissions of the extension can be set in the manifest. For the purpose of testing the extension framework on a locally hosted webpage, the permissions are set to *<all_urls>*.

4.3 User Interface

The user interface displays one button to turn off detection.

The extensions popup is the sole user interface, offering a single switch button to enable or disable detection for all tabs throughout the current browser session. This status is consistently stored for subsequent browser sessions using the *chrome.storage API*.
It is worth noting that when deactivating detection, the popup script uses a simple approach by reloading the page, undoing any prior modifications made to the webpage.

4.4 Service Worker

The service worker handles background processes.

The service worker acts as the background script in MV3. It orchestrates the data flow and ensures that scripts are executed in the correct order. While each tab has its own

[2]https://developer.chrome.com/docs/extensions/reference/api/storage
Accessed: March 2024

version of the content script, the service worker remains constant.
When a request for the analysis and application of countermeasures in a tab is indicated by the popup (see Section 4.3) or the content script (see Section 4.5), the service worker requests the execution of certain actions within the content script, which are described in Section 4.5.2.
Once these actions have been performed, the service worker receives a response from the content script, including HTML and CSS content of the webpage (see Figure 3.1). Immediately afterwards, the service worker sends a message to the *Detector* to initiate the detection process (4.6.1). This message includes the necessary information for the detection process, using the JSON schema shown in Listing 4.1. The keys *html* and *css* encode the HTML and CSS content, and the *tabId* key encodes the tab id this data belongs to. The tab id can easily be read from the metadata of the content script response.

The corresponding tab ID, along with HTML and CSS content is sent to the *Detector*.

As illustrated in Figure 4.1, the service worker receives the detection data from the *Detector* and forwards it to the *Converter*. The detection data format is discussed in Section 4.6.1. The countermeasure process takes place in the *Converter* (see Section 4.6.2), and the service worker expects a response message including the countermeasure data, as described in Section 4.5.3.

The detection data is forwarded to the next module.

It is crucial that the changes requested by the *Converter*, are applied in the correct tab. This is ensured because the service worker not only sends the tab ID to the *Detector* (as mentioned earlier) but also to the *Converter*. It also expects the tab ID to be forwarded back to the service worker within the payload of each response, in order to guarantee the correct assignment at every point until injection.

The tab ID is used as a message identifier.

The service worker is responsible for one requirement of the **communication with the DOM**, as outlined in the requirement analysis in Section 3.1: the injection of a custom stylesheet. This stylesheet that can be used by the *Converter* to add new CSS classes. Therefore it is carried out right before the content script is tasked with injecting the DOM changes as requested by the *Converter*.
The stylesheet is an asset, customisable by the *Converter* module, where CSS style classes can be defined statically to

Listing 4.1: JSON schema - input for detection

```
{
    "$schema": "https://json-schema.org/draft/2019-09/
       schema",
    "title": "Webpage Content Schema",
    "description": "Schema for representing HTML and CSS
       content of a webpage along with its tab identifier.
       ",
    "type": "object",
    "properties": {
        "tabId": {
            "description": "The unique identifier of the
               browser tab.",
            "type": "string",
            "pattern": "^[0-9]+$"
        },
        "html": {
            "description": "The HTML content of the
               webpage, including any inline data
               attributes.",
            "type": "string",
        },
        "css": {
            "description": "The CSS styles associated with
                the webpage.",
            "type": "string",
        }
    },
    "required": ["tabId", "html", "css"],
    "additionalProperties": false
}
```

facilitate the countermeasure process. The service worker itself is unaware of the files content, as it is only important for the *Converter* and can not be of harm if used incorrectly by the *Converter* module developer.
Listing 4.2 shows an example of a CSS class, that is used in our proof of concept (see Section 5.1), when highlighting two elements together like in Figure 2.2.
A new div-element would be added to the webpage, enclosing the to be highlighted elements. Then, the class

red-border (as in Listing 4.2) that was defined inside of the stylesheet beforehand, would be added to the div-element to change among others the border colour of this element.
Following the injection of the stylesheet, the content script is tasked to inject the actual countermeasures. There is an instance of the content script for each tab individually and these instances have to be addressed with their respective tab ID. Therefore, the service worker deletes the property *tabId* from the countermeasure data and instead uses its value within to trigger the correct content script (see 4.5.3).

The countermeasuring data is forwarded to the content script.

Listing 4.2: CSS classes example

```
.red-border {
    border: 2px dashed red;
    display: inline-block;
    padding: 5px;
}
```

4.5 Content Script

The content script is responsible for any communication with the DOM and is hence one of the most intricate components. It handles the observation of changes in the DOM, which is described in Section 4.5.1.
This implementation also covers the first set of requirements for the **communication with the DOM**, as outlined in the requirement analysis in Section 3.1: Extracting HTML and CSS contents. The implementation of these actions, plus the injection of unique identifiers to each HTML element, corresponds to the requirement of enabling precise referencing of individual elements under the topic **data formats for communication between scripts** in Section 3.1 (discussed in Section 4.5.2).
In Section 4.5.3 the injection of changes in the DOM by the content script, based on information sent by the *Converter* is presented.

The content script injects element IDs and extracts the website's contents at the beginning of the process, and injects changes at the end of the process.

4.5.1 Detecting DOM Changes

When changes occur on the webpage, for instance when a webpage is opened/reloaded or elements are changed dynamically, the content script has the ability to monitor these changes and restart the detection process. By utilising the MutationObserver[3] interface, the script listens for any additions, removals and other alterations within the webpage's structure. It aims to consider altered or new elements, that could potentially harbour DPs.
The extension is however not operating like the implementation by Hausner and Gertz, the *Dapde Pattern Highlighter*, which takes two screenshots of the DOM within an interval of 1.5 seconds and analyses both copies in relation to another to detect patterns that rely on dynamic changes, such as countdowns.

A debounced callback function is implemented to balance responsiveness and performance, preventing excessive resource consumption on pages with frequent updates. However, this function could possibly be improved after testing on more than the test webpage, in order to find right balance (see Section 5).
Additionally it is ensured that no observation takes place during the injection,which would lead to an endless cycle.

4.5.2 Extracting Webpage Contents & Injecting IDs

Upon receiving a request to extract the webpage contents from the service worker (see Section 4.4), the content script begins its first phase of responsibilities.
As already mentioned in the requirement analysis in Section 3.1, it is necessary to extract the HTML and CSS content from the respective tab, for a comprehensive detection. But more importantly, before reading this data, a unique ID for each HTML element is injected into the

[3]https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver
Accessed: March 2024

Before extracting data, element IDs are injected into the DOM.

DOM. The modules must be able to reference individual HTML elements in the DOM to identify elements that are part of a DP and to pinpoint which elements should be altered. In this implementation, unique IDs are created by the content script and added in the form of a dataset property[4], namely *DecDefId*. Dataset properties allow data to be stored directly within HTML elements without interfering with the page's presentation or behaviour.
The decision to inject the property *DecDefId* before the reading process is convenient because the HTML contents can be read right afterwards, all IDs are already included in this data and accessible for the *Detector* and *Converter*. Furthermore, the IDs (*decDefIds*) are already present in the DOM and can be used by the content script during injection.
After the injection of the IDs is concluded, the content script extracts HTML and CSS content from the current context and it is sent back to the service worker.

4.5.3 Injecting Visual Countermeasures

The final responsibility of the content script, as a part of the extension framework, is to implement requested alterations of the webpage. The main objective of this implementation is to equip the *Converter* with all the necessary features specified in the requirement analysis (see Section 3.1). The required features include changes to the styles and attributes of existing elements, as well as the removal and addition of elements.

The requested countermeasures, encoded with the JSON format, are applied by the content script.

The JSON schema in Listing A.1 provides a structured blueprint for the countermeasure data sent from the *Converter* to the service worker, dictating the modifications to be implemented within the webpage's DOM. As mentioned previously in Section 4.4, the property *tabId* is part of the payload for every communication between the modules and the service worker, but is not required as a

[4] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement *Accessed: March 2024*

part of the countermeasure data that is sent to the content script for injection. Therefore, the content script receives the information in a format shown in A.1, but without the property *tabId*. As soon as this happens, the process of injection is initiated.

Encoded within the countermeasuring data are: elements to add, elements to delete, changed styles and attributes.

The property *addElements* in Listing A.1 can hold a list of items, each representing a new element to be created and added to the DOM. This functionality is necessary for example when adding an element that groups elements together, to apply a countermeasure such as HL+E by Schäfer et al. [2023], shown in Figure 2.2, using a red dotted line for highlighting.

New elements aree encoded using: anchor element, type, attributes, styles and placement

Each element to be added is characterised by their anchor element, type, attributes, styles and placement (encoded in the properties *anchorElementId, type, styles, attributes* and *placement*). The name of the first property implies that it serves as an anchoring point, to place the newly created element using the property value of *placement* relative to it. The *placement* keyword accepts four values: *afterend, beforebegin, beforeend* & *child*. These options are chosen to cover all potentially required placement options. This involves adding parent elements (cloning the parent and adding it as a child) or children to existing elements, but also to elements that were added previously in the same process. Moreover, it can be achieved to add child elements in a specific order, using *afterend*, *beforebegin* or *beforeend* with another child element as an anchoring point.
The anchor element itself could be either already part of the DOM, or is created and added previously in the same countermeasure process and referenced in the JSON format by its *decDefId*. When using the second approach, the *decDefId* can be added as an attribute using the keyword *data-dec-def-id*.
The value of *type* is used to define the type of element to be created, such as a div-element or a span-element.
Lastly, the properties *styles* and *attributes* can feature a list of modified styles or attributes to be changed after creation, using for instance the key-value pair *(display, inline-block)* as a style modification or *(textcontent, "Some text")* as an attribute modification.
Entire CSS classes, like the ones shown in Listing 4.2, can be

CSS classes can be used to group styles.

added or removed within the property *styles*. The content script scans for the key *classList.add* or *classList.remove* and simply adds or removes the corresponding value as a class of the respective element.

To alter existing elements, their ID along with new styles and attributes are encoded.

The next main property *stylesAndAttributes*, holds a list of items where each item encloses the properties *elementId*, *styles* and *attributes*. The *decDefId* of the element, encoded in *elementId*, refers to an existing element of the webpage. The *Converter* can encode styles and attributes to be modified on the corresponding element within the keywords *styles* and *attributes* the same way as with the newly created elements.

Elements are deleted, using their ID.

The last property of the format is *deleteElements*, consisting just of a list of element IDs (*decDefIds*). The corresponding elements to the encoded IDs will be deleted from the webpage.

4.6 Modules

The extension framework provides two interfaces for the integration of a module to detect DPs (the *Detector*) and a module for applying countermeasures (*Converter*), to complete this framework. These modules are intended to be exchangeable by other developers through their respective approaches. In the course of this thesis, both modules are hard coded for the purpose of testing the extension framework on a test webpage. The requirements for the *Detector* and *Converter* are discussed in the following subsections.

4.6.1 Detector

DP instances are detected based on HTML and CSS contents by the *Detector*.

The *Detector* module is the first in line, being provided with the extracted webpage contents by the service worker. In essence, the *Detector* should consider all available information to determine which elements of the HTML

constitute a DP. The HTML and CSS provides a comprehensive database for detection, as discussed in Section 3.1, covering among others information about the six essential properties for detection identified by Chen et al. [2023].
The IDs (*DecDefId*) that were assigned earlier by the content script (see Section 4.5.2) can be used to pinpoint the detected elements to the service worker.

The *Detector* requires a listener and a response for the service worker's messages.

The implementation of this module only requires a listener to receive requests by the service worker as well as a response with detection data as shown in Figure 4.1.
This response is extended by the service worker before forwarding it to the countermeasure module. Additionally, the HTML and CSS content that was sent to the *Detector* is added too, fulfilling the requirements outlined in Section 3.1. The *Detector* must not send the tabID back to the service worker, because the response is sent in the same messaging channel. Considering the deliberately low-kept requirements for communication in-between the modules, we use the JSON schema from Listing A.2. Additionally to encoding the tab ID within the property *tabId*, there is another property *pattern* which holds a list of items, each constituting one element on the page. The two main properties of each item are *elementId* and *patternId* to pinpoint the specific element on the webpage and provide information about which DP category it belongs to, corresponding to the requirements outlined in Section 3.1.
Since single elements of a web page are rarely a DP in isolation [Hausner and Gertz, 2021], there is an optional property *related* which lists IDs of elements that belong to the same DP instance.

4.6.2 Converter

Detection data is converted into countermeasuring data.

The job of the *Converter* module is to choose countermeasure strategies tailored to the detected DP instances by the *Detector*. More specifically, it decides which styles and attributes of which element should be altered, and whether to add new elements or to delete some.

The *Converter* requires a listener and a response for the service worker's messages.

Like the *Detector*, the *Converter* requires a listener to receive

requests by the service worker as well as a response with conversion data as shown in Figure 4.1.
The *Converter* receives the detection data from the *Detector*, which was extended from the service worker by HTML and CSS content, as explained in Section 4.4. Access to this information enables the *Converter* to consider the original state and context of each detected element and choose appropriate alterations.
In Section 4.6.1 the format for the detection data was introduced. The *Converter* module must adapt to this JSON schema.

The countermeasure data has to be encoded using the framework's JSON format.

Additionally, the *Converter* has to meet the required encoding format for the response to the service worker, as described previously in Section 4.5.3.

The prior injection of a custom stylesheet by the service worker (see Section 4.4), allows the *Converter* to use CSS classes for grouping related style changes together. This way, the module can communicate the injection of entire blocks of style changes to the extension framework, by encoding the class key within a property inside of *styles* within the JSON schema A.1. Using for instance the key-value pair *(classList.add, red-border)* as a property of the keyword *styles*, the class *red-border* shown in Listing 4.2 can be applied.

Chapter 5

Evaluation & Discussion

5.1 Proof of Concept

As a final step, the extension framework was tested with hard-coded modules on a website that was solely created for this purpose. Hard-coded means in this case, that the modules only implement the necessary listeners and response mechanisms as outlined in Section 4.6.1 and 4.6.2, but there is no implementation of an algorithm for detection or for applying countermeasures.

The countermeasured HL+E, LL and SW were implemented testwise.

The countermeasures HL+E, LL and SW were implemented test-wise on the locally hosted webpage. As outlined in the requirement analysis from Chapter 3.1, these countermeasures reflect all necessary functions for the implementation of the remaining designs for countermeasures tested by Schäfer et al. [2023].
It was possible to showcase the three countermeasure designs on three DP types: *Confirmshaming*, *Visual Interference* and *Low-stock Message*. We chose them, because these DP types were the ones tested in the user study by Schäfer et al. [2023]. Figure 5.1 shows the unaltered instance of *Confirmshaming* that was built into the test webpage,and three replicas of this design, with countermeasures applied on this instance. For the countermeasure HL+E, the *Converter* module requests a modification of the border colour of the element that encloses both buttons. Additionally, a new

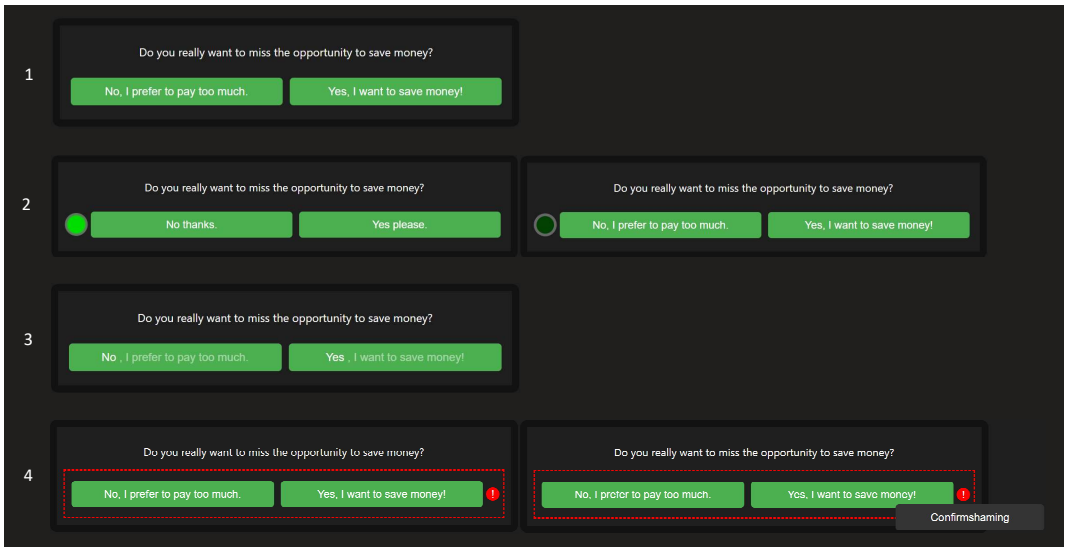

Figure 5.1: The countermeasures Highlight with Explanation (HL+E), Lowlight (LL) and Switch (SW) applied on a test webpage using the *Deceptive Defender* framework and hard-coded modules.
1) Unchanged DP instance 2) Countermeasure SW
3) Countermeasure LL 4) Countermeasure HL+E

icon element is added to the webpage, which displays a popup when hovering over it, as shown in Figure 5.1. The next DP instance is counter measured with LL. We added each two child elements to the button, with the first element just displaying the content *yes* and *no* in their original colour, and the second element displaying the remaining text contents that were detected to be with malicious intent, slightly less visible.
For the countermeasure SW, a more a more complex alteration had to be made. Firstly, the countermeasure HD was applied on the detected elements. This is done by removing or hiding the text contents that were changed in color or transparency in the previous countermeasure. Secondly, a new button item had to be created, which should provide the functionality to switch back to the original state of the altered elements. It was possible to integrate this function by only using CSS classes from the custom stylesheet that was injected by the service worker.

5.2 Discussion

The *Deceptive Defender* framework is a technical intervention measure with varying intervention scopes.

The *Deceptive Defender* framework aligns with the category of technical intervention measures by Bongard-Blanchy et al. [2021], which includes practical tools, developed to disrupt or mitigate the effects of DPs. The authors categorised tools, similar to existing extensions like *Insite* or the *Dapde Pattern Highlighter*, in the intervention measure DP Elimination. However, this framework can not be assigned to only one of the intervention scopes created by Bongard-Blanchy et al. [2021], since the scope is set by the implemented countermeasures. Visual countermeasures align with the design intervention measure and can correspond to more than one intervention scope. The HL+E countermeasure by Schäfer et al. [2023] enhances user awareness, not only highlighting the manipulative elements but also providing explanatory notes that inform the user about the nature of the deception. This design aligns with the DP Awareness or DP Detection scope, trying to enhance user's ability to recognize this DP type in the future.
The countermeasure HD on the other hand, seeks to eliminate the DP entirely from the user's view [Schäfer et al., 2023]. This intervention falls within the DP Elimination scope, aiming to remove manipulative elements so that they have no impact on user behaviour.

The modular approach harbours restrictions, but more freedom on other ends.

In Section 3, we outlined the key advantages kept in mind when designing this modular framework: flexible testing of new detection and countermeasure algorithms and future proofing through an adaptable set of covered DP types.
Considering the current state of research in detection algorithms and visual countermeasures, outlined in Section 2, the *Deceptive Defender* framework aims to establish a basis for the HCI community to trial novel methods. In Section 3, we highlighted the flexibility of the *Deceptive Defender* framework, particularly its ability to adapt to new types of DPs without altering the core system. While maintaining the JSON schemas that are used on the interfaces of this framework might appear restrictive, we believe this disad-

vantage doesn't weigh out the advantage of the developer of the detection or countermeasure module being able to focus more on developing corresponding solutions and less on developing the extension's fundamental components.

During this project, we encountered some challenges and questions regarding the independence of the *Detector* from the *Converter* module, which is one of the main focusses in this work.

Our data format for the detection data is a double-edged sword.

Firstly, we want to emphasise that the *Detector* is the bottle neck of the extension. No matter how many advanced and specialised countermeasures the *Converter* implements, he can not use them when the detection method is not as advanced.
Additionally, our chosen data format for the detection data (see Listing A.2) is a double-edged sword. It offers the categorisation of detected elements into DP types, which means that every kind of taxonomy can be used and the number of DP types is not limited. This is an advantage because this project aimed at providing the opportunity to detect any kind of DP. On the other hand, the *Converter* module has to be adapted to this encoding, in the sense that the keywords used by the *Detector* as the value of the key *patternId* lead to the application of the correct countermeasure by the *Converter*. This required 'common ground' regarding the DP categorisation restricts the independence from the *Converter*.

There is a need to conduct how fine-grained the detection has to be for the *new* countermeasures introduced by Schäfer et al. [2023].

The design of a communication schema in-between the modules was challenging in general, while not being able to comprehend how fine-tuned the detection has to be in order to apply the countermeasures by Schäfer et al. [2023] properly. For example, the question came up, whether it is the *Converter's* or the *Detector's* responsibilities to detect which part of the DP a specific element constitutes. When there is a DP instance which consists of more than one element, such as an instance of *Visual Interference* with differently coloured *accept*- and *deny* buttons, existing browser extensions such as the *Dapde Pattern Highlighter* would just highlight these buttons. However when using other countermeasures, there may be the need for more fine-grained information about the structure of the DP instance. For example, when using the countermeasure HD by Schäfer

Listing 5.1: Example of a JSON encoding strategy for relationships between elements

```
{
  "patterns": [
    {
      "elementId": "0"
      "patternId": "visualInterferenceAccept",
      "related": ["denyButton"],
    },
    {
      "elementId": "1"
      "patternId": "visualInterferenceDeny",
      "related": ["acceptButton"],
    }
  ]
}
```

et al. [2023] on the described DP instance, the *Converter* would require information about which element is the *accept* button and which is the *deny* button, to change the color of the *accept* button to match that of the *deny* button. In the JSON schema for communication between *Detector* and *Converter* (see Listing A.2) this information could be actually encoded inside of the *patternId* property within the *patterns* property. For instance these described *accept*- and *deny* buttons could be encoded as shown in Listing 5.1, using the keywords *visualInterferenceAccept* and *visualInterferenceDeny* to encode what role they take on in the DP. Again, here arises the problem of the *Converter* loosing it's independence from the *Detector* module. In future work, for instance when designing these modules, it has to be evaluated which information is required for the *Converter* to apply specific countermeasure styles. Based on that, a more advanced communication in-between the modules may be developed.

Apart from that, the proof of concept showed that it is possible to implement the countermeasures SW, LL and HL+E on each three different DP instances. Even when not for the purpose of this extension framework, it was proven

that changes on the DOM, constituting these countermeasures by Schäfer et al. [2023], can be encoded within the proposed JSON schema by the *Converter* and systematically read and applied by this framework. Especially the switch function could be implemented by just using CSS styles instead of javascript functions.

Notably, when deactivating detection in this framework, the popup script employs a straightforward approach by reloading the page in order to revert prior modifications. This approach is inconvenient for the user, because it is time consuming and information can get lost. The *Dapde Pattern Highlighter* shows how to implement a more fine-tuned approach, by adding CSS classes to elements associated with a DP, and removing those classes again when the deactivation is requested.

Adapting the strategy of the *Dapde Pattern Highlighter* for reversing applied countermeasures is more complex within the context of our extension.

Adapting this strategy within this extension framework presents challenges, because in contrast to the *Dapde Pattern Highlighter*, our extension framework provides a wider spectrum of countermeasure tools: Instead of using exactly two predefined CSS classes that are added to each detected DP instance, the *Deceptive Defender* framework enables the implementation of any number of CSS classes and single style and attribute changes. Subsequently it is more complex to restore the original state of the webpage.
This example showcases, how the modular design of this extension complicates implementation, which may lead to the necessity of further development of the framework, which is elaborated in Section 6.2.

Chapter 6

Summary & Future Work

In this chapter, we conclude this project by providing a summary of our work and outlining opportunities for future work.

6.1 Summary

This thesis explored the design of a modular browser extension for applying visual countermeasures in the web. The goal was to build a foundation for easy integration of different algorithms for automatic pattern detection and applying countermeasures. The proof of this concept process was accomplished in this thesis in three steps:

We outlined the requirements for the implementation of *Deceptive Defender*.

Firstly, we established the groundwork by defining the functional and non-functional requirements critical to the development of such an extension framework. The requirements for the implementation of countermeasures were based on the countermeasures by Schäfer et al. [2023], covering addition and deletion of elements as well as changing style and attribute properties of existing elements. Also outlined were requirements regarding

the communication between the extension framework and the exchangeable modules, a method to reference particular elements in the webpage, and to ensure the correct assignment of detection and countermeasure data to the corresponding tab.

We documented each step of the implementation process.

Secondly, we provided a brief overview of the framework's architecture and the key responsibilities of its components, including the modules. Chapter 4 documents each step of the implementation, detailing decisions regarding timing, responsibility distribution and communication formats. The JSON schemes for communication from the service worker to the *Detector* (see 4.4) as well as from the *Converter* (see 4.5.3) back to the service worker are set, while the communication format between the *Detector* and the *Converter* is partly customisable by the module developer. The extension framework has one content script that features different injection functionalities to the *Converter* module, that cover the required actions outlined in the first step, the requirement analysis.

We performed a proof of concept by implementing three countermeasures.

Lastly, a webpage was created to implement the malicious design patterns *Confirmshaming*, *Visual Interference* and *Low-stock Message*. Specifically these DPs were used because we based our framework on the work of Schäfer et al. [2023], using the same DPs as them in their study. This webpage served as a prototype to test the extension framework*Deceptive Defender* in combination with hard-coded modules for the time being. The implementation of three selected countermeasures on each of the DP instances, although limited in scope, constituted a proof of concept.

6.2 Future Work

Testing was performed only with hard-coded modules and a hard-coded website.

One limitation is that we tested the extension framework only with hard-coded modules and on a webpage that contains no dynamic changes. When developing the first detection and countermeasure module, it may be necessary

to refine the frameworks implementation. That especially involves the observation of the DOM through the content script. This observation must be well balanced in order to prevent excessive demands but to initiate a new detection process when necessary. Mathur et al. [2019] uses the Mutation Summary[1] library to observe changes in the DOM during a crawl, stated by the authors of the library to be useful especially for browser extensions.

The development and integration of modules is necessary.

Absolutely critical is the development of the *Detector* and *Converter* modules. After implementing and integrating the required modules, the extension's effectiveness in a real-world context can be fully assessed. There is also the possibility to implement solely the *Converter* module based on a hard-coded webpage (such as our test website) and a hard-coded *Detector* module. With this configuration, countermeasure algorithms are testable. Additionally, user studies on countermeasure designs could be conducted in a more realistic setting than compared to the study by Schäfer et al. [2023]. This can be done either with the mentioned configuration, solely implementing the *Converter* and a hard-coded *Detector*, or the implementation of both modules and the use of real websites. Both ways, valuable insights into their practical impact, user acceptance, and areas for improvement can be gained.

Requirements of the *Converter* must be investigated more closely.

As elaborated in Section 5.1, the communication in-between the modules has weaknesses, making the *Converter* lose part of it's independence from the *Detector*. I propose, that the requirements of the *Converter* for applying specific countermeasures have to be investigated more closely. Based on that, a more advanced communication system in-between the modules may be developed.

Possible improvement: More functionalities on the extension's popup.

A potential future improvement of the framework itself is enabling the user to activate the countermeasures only selectively across multiple tabs or webpages. Additionally, users might benefit from the capability to toggle specific countermeasures based on their preferences, since the user

[1] https://github.com/rafaelw/mutation-summary *Accessed: March 2024*

study by Schäfer et al. [2023] indicated that user requirements may change when their expertise level increases and some countermeasures could be perceived as annoying over time.

An additional improvement opportunity for the framework lays in the process of reverting the applied countermeasures. Currently, the framework employs a straightforward approach that was elaborated and discussed in Section 5.2. This procedure could be improved, for instance by adapting a similar approach as the *Dapde Pattern Highlighter* as described in Section 5.2.

Appendix A

JSON schema

In the following, the JSON schema that is expected by the extension frameworks content script and therefore to be implemented as the schema for outgoing responses by the *Converter*, as well as the JSON schema for the communication between the detection and countermeasuring module are listed.

Listing A.1: JSON schema - output of *Converter*

```
{
"$schema": "https://json-schema.org/draft/2019-09/schema",
"title": "Message from the Converter module to the service worker",
"properties": {
  "tabId": {
    "description": "Destination tab ID",
    "type": "string"
  },
  "addElements": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "anchorElementId": {
          "description": "Element ID of the anchor element",
          "type": "string"
        },
        "type": {
          "description": "Type of the object for creation",
          "type": "string"
        },
        "attributes": {
          "description": "List of modified attributes for the new element",
          "type": "object",
          "additionalProperties": {
            "description": "Specific attribute modification on the new element",
            "type": "string"
          }
        },
```

```
38          "styles": {
39            "description": "List of modified styles for the
                 new element",
40            "type": "object",
41            "additionalProperties": {
42              "description": "Specific style modification on
                   the new element",
43              "type": "string"
44            }
45          },
46          "placement": {
47            "description": "Placement of the element for
                 creation, relative to the defined anchor
                 element",
48            "type": "string",
49            "enum": ["afterend", "beforebegin", "beforeend",
                  "child"]
50          }
51        },
52        "required": ["anchorElementId", "type", "placement"]
53      }
54    },
55    "stylesAndAttributes": {
56      "type": "array",
57      "items": {
58        "type": "object",
59        "properties": {
60          "elementId": {
61            "description": "Element ID of the element to
                 undergo modification",
62            "type": "string"
63          },
64          "attributes": {
65            "description": "List of modified attributes for
                 the element",
66            "type": "object",
67            "additionalProperties": {
68              "description": "Specific attribute
                   modification on the element",
69              "type": "string"
70            }
71          },
```

```
76          "styles": {
77            "description": "List of modified attributes for
                 the element",
78            "type": "object",
79            "additionalProperties": {
80              "description": "Specific style modification on
                    the element",
81              "type": "string"
82            }
83          }
84        },
85        "required": ["elementId"]
86      }
87    },
88    "deleteElements": {
89      "description": "List of elements for deletion",
90      "type": "array",
91      "items": {
92        "description": "Element ID of the element for
             deletion",
93        "type": "string"
94      }
95    }
96  },
97  "additionalProperties": false
98  }
```

Listing A.2: JSON schema - output of *Detector*

```
{
    "$schema": "https://json-schema.org/draft/2019-09/
       schema",
    "title": "Message from the Detector module to the
       service worker",
    "type": "object",
    "properties": {
        "patterns": {
            "description": "List of detected patterns with
               their related elements",
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "patternId": {
                        "description": "Identifier of the
                           detected pattern",
                        "type": "string"
                    },
                    "elementId": {
                        "description": "Element ID
                           associated with the detected
                           pattern",
                        "type": "string"
                    }
                },
                "required": ["patternId", "elementId"]
            }
        }
    },
    "required": ["tabId", "patterns"],
    "additionalProperties": false
}
```

Bibliography

V. Aravind and M. Sethumadhavan. A framework for analysing the security of chrome extensions. In M. K. Kundu, D. P. Mohapatra, A. Konar, and A. Chakraborty, editors, *Advanced Computing, Networking and Informatics-Volume 2*, pages 267–272, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07350-7.

K. Bongard-Blanchy, A. Rossi, S. Rivas, S. Doublet, V. Koenig, and G. Lenzini. "i am definitely manipulated, even when i am aware of it. it's ridiculous!" - dark patterns from the end-user perspective. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference*, DIS '21, page 763–776, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384766. doi: 10.1145/3461778.3462086. URL https://doi.org/10.1145/3461778.3462086.

C. Bösch, B. Erb, F. Kargl, H. Kopp, and S. Pfattheicher. Tales from the dark side: Privacy dark strategies and privacy dark patterns. *Proceedings on Privacy Enhancing Technologies*, 2016:237–254, 07 2016. doi: 10.1515/popets-2016-0038.

J. Chen, J. Sun, S. Feng, Z. Xing, Q. Lu, X. Xu, and C. Chen. Unveiling the tricks: Automated detection of dark patterns in mobile applications, 2023.

C. Cockburn and T. D. Wilson. Business use of the world-wide web. *Inf. Res.*, 1, 1996. URL https://api.semanticscholar.org/CorpusID:43010893.

European Commission, Directorate-General for Justice, Consumers, F. Lupiáñez-Villanueva, A. Boluda, F. Bogliacino, G. Liva, L. Lechardoy, and T. Rodríguez

de las Heras Ballell. *Behavioural study on unfair commercial practices in the digital environment – Dark patterns and manipulative personalisation – Final report*. Publications Office of the European Union, 2022. doi: doi/10.2838/859030.

G. Conti and E. Sobiesk. Malicious interface design: Exploiting the user. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 271–280, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772719. URL https://doi.org/10.1145/1772690.1772719.

R. Coudert. Automatically detect dark patterns in cookie banners. Master's thesis, École Polytechnique Fédérale de Lausanne (EPFL), SPRING Lab, Computer Science Department, Grenoble, France; Lausanne, Switzerland, August 2020.

A. Curley, D. O'Sullivan, D. Gordon, B. Tierney, and I. Stavrakakis. The design of a framework for the detection of web-based dark patterns. In *Proceedings of the 15th International Conference on Digital Society (ICDS)*. Technological University Dublin, 7 2021. Online conference.

L. Festinger. *A Theory of Cognitive Dissonance*. Stanford University Press, Redwood City, 1957. ISBN 9781503620766. doi: 10.1515/9781503620766. URL https://doi.org/10.1515/9781503620766.

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*, CHI '18, page 1–14, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3174108. URL https://doi.org/10.1145/3173574.3174108.

C. M. Gray, L. Sanchez Chamorro, I. Obi, and J. Duane. Mapping the landscape of dark patterns scholarship: A systematic literature review. In *Companion Publication of the 2023 ACM Designing Interactive Systems Conference*, DIS '23 Companion, page 188–193, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9781450398985. doi: 10.1145/3563703.3596635. URL https://doi.org/10.1145/3563703.3596635.

C. M. Gray, C. Santos, and N. Bielova. Towards a preliminary ontology of dark patterns knowledge. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023b. Association for Computing Machinery. ISBN 9781450394222. doi: 10.1145/3544549.3585676. URL https://doi.org/10.1145/3544549.3585676.

S. Greenberg, S. Boring, J. Vermeulen, and J. Dostal. Dark patterns in proxemic interactions: A critical perspective. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, page 523–532, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329026. doi: 10.1145/2598510.2598541. URL https://doi.org/10.1145/2598510.2598541.

P. Hausner and M. Gertz. Dark patterns in the interaction with cookie banners. *CoRR*, abs/2103.14956, 2021. URL https://ds.ifi.uni-heidelberg.de/files/Team/phausner/publications/Hausner_Gertz_CHI2021.pdf.

D. Kahneman. A perspective on judgment and choice: mapping bounded rationality. *The American psychologist*, 58 9:697–720, 2003. URL https://api.semanticscholar.org/CorpusID:16994141.

C. Lacey and A. Beattie. Clusters of dark patterns across popular websites in new zealand. 2023. URL https://api.semanticscholar.org/CorpusID:261077946.

K. Lukoff, A. Hiniker, C. M. Gray, A. Mathur, and S. S. Chivukula. What can chi do about dark patterns? In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI EA '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380959. doi: 10.1145/3411763.3441360. URL https://doi.org/10.1145/3411763.3441360.

A. Mathur, G. Acar, M. J. Friedman, E. Lucherini, J. Mayer, M. Chetty, and A. Narayanan. Dark patterns at scale: Findings from a crawl of 11k shopping websites. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), nov 2019. doi: 10.1145/3359183. URL https://doi.org/10.1145/3359183.

A. Mathur, M. Kshirsagar, and J. Mayer. What makes a dark pattern... dark? design attributes, normative considerations, and measurement methods. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445610. URL https://doi.org/10.1145/3411764.3445610.

M. Nouwens, I. Liccardi, M. Veale, D. Karger, and L. Kagal. Dark patterns after the gdpr: Scraping consent pop-ups and demonstrating their influence. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20. ACM, April 2020. doi: 10.1145/3313831.3376321. URL http://dx.doi.org/10.1145/3313831.3376321.

C. P. Ross. What is the internet. *Seg Technical Program Expanded Abstracts*, pages 1537–1537, 1995. URL https://api.semanticscholar.org/CorpusID:140121659.

R. Schäfer, P. M. Preuschoff, and J. Borchers. Investigating visual countermeasures against dark patterns in user interfaces. In *Proceedings of Mensch Und Computer 2023*, MuC '23, page 161–172, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400707711. doi: 10.1145/3603555.3603563. URL https://doi.org/10.1145/3603555.3603563.

T. H. Soe, C. T. Santos, and M. Slavkovik. Automated detection of dark patterns in cookie banners: how to do it poorly and why it is hard to do it any other way, 2022.

C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz. (un)informed consent: Studying gdpr consent notices in the field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19. ACM, November 2019. doi: 10.1145/3319535.3354212. URL http://dx.doi.org/10.1145/3319535.3354212.

A. E. Waldman. Cognitive biases, dark patterns, and the 'privacy paradox'. *Current Opinion in Psychology*, 31:105–109, 2020. ISSN 2352-250X. doi:

https://doi.org/10.1016/j.copsyc.2019.08.025. URL https://www.sciencedirect.com/science/article/pii/S2352250X19301484. Privacy and Disclosure, Online and in Social Interactions.

Index

Typeset April 10, 2024