DarkFleece: Probing the Dark Side of Android Subscription Apps

Chang Yue[1,2], Chen Zhong[3], Kai Chen[1,2]*, Zhiyu Zhang[1,2], and Yeonjoon Lee[4]*

[1]Institute of Information Engineering, Chinese Academy of Sciences, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, China
[3]University of Tampa, USA
[4]Hanyang University, Ansan, Republic of Korea
{yuechang, chenkai, zhangzhiyu1999}@iie.ac.cn, czhong@ut.edu, yeonjoonlee@hanyang.ac.kr

Abstract

Fleeceware, a novel category of malicious subscription apps, is increasingly tricking users into expensive subscriptions, leading to substantial financial consequences. These apps' ambiguous nature, closely resembling legitimate subscription apps, complicates their detection in app markets. To address this, our study aims to devise an automated method, named DarkFleece, to identify fleeceware through their prevalent use of dark patterns. By recruiting domain experts, we curated the first-ever fleeceware feature library, based on dark patterns extracted from user interfaces (UI). A unique extraction method, which integrates UI elements, layout, and multifaceted extraction rules, has been developed. DarkFleece boasts a detection accuracy of 93.43% on our dataset and utilizes Explainable Artificial Intelligence (XAI) to present user-friendly alerts about potential fleeceware risks. When deployed to assess Google Play's app landscape, DarkFleece examined 13,597 apps and identified an alarming 75.21% of 589 subscription apps that displayed different levels of fleeceware, totaling around 5 billion downloads. Our results are consistent with user reviews on Google Play. Our detailed exploration into the implications of our results for ethical app developers, app users, and app market regulators provides crucial insights for different stakeholders. This underscores the need for proactive measures against the rise of fleeceware.

1 Introduction

The integration of subscription models into mobile apps has witnessed significant growth in recent years. Subscription revenue has grown from around $1.5 billion in 2015 to $17.1 billion in 2022 [12]. However, the popularity of subscriptions has led to an increase in their abuse. Deceptive mobile apps, referred to as **fleeceware** [52], have been a pervasive issue in the mobile app market. These apps incorporate dark design patterns, such as unclear terms, to deceive users into subscribing to overly expensive services. Users may subscribe to apps without their consent, fail to realize they will be charged after a free trial, or struggle to cancel a subscription. Recent reports have highlighted subscription scams [14, 30, 47, 50, 58]. According to Avast, 69 fleeceware apps generated $38.5 million in revenue in 2019 [4].

Given its widespread prevalence, automated detection of fleeceware is imperative to minimize its harmful effects. However, these apps typically do not contain malicious code and exhibit coding patterns that closely resemble legitimate apps, making it difficult to detect them using traditional methods such as malicious code detection. Presently, app markets largely lean on monetization requirements as a regulatory tool to combat fleeceware [28]. However, due to the absence of automated detection methods, the process is manual, which is neither timely nor scalable. This work is dedicated to investigating fleeceware at scale through the development of an automatic fleeceware detection system.

Current observations reveal that fleeceware often employs dark patterns [10, 50], which are deceptive interface designs intended to manipulate users into paying subscription fees beyond their initial intention. Therefore, we mainly focus on identifying these dark patterns in UIs for fleeceware detection. An app with dark pattern features doesn't automatically qualify as fleeceware, so we utilize a model that combines these features for detection. Our results are presented in a user-friendly manner, making it easy for users to understand.

To effectively pinpoint fleeceware, we developed DarkFleece, a Dark Pattern-based Fleeceware Detector, which meticulously scrutinizes the UI design of subscription-based apps to uncover any embedded dark patterns. To realize DarkFleece, we mainly addressed two challenges.

C1: Constructing a Fleeceware Feature Library. Implementing an automatic fleeceware detection system requires recognizing the distinct features of fleeceware. There is a lack of research on identifying detectable features that can accurately describe fleeceware, and the availability of a well-labeled dataset for fleeceware is limited. Despite extensive research on dark patterns, the automated detection of fleece-

*The corresponding authors.

ware remains elusive due to the complexities and variabilities intrinsic to fleeceware patterns. Drawing a clear line between fleeceware and legitimate apps is far from straightforward.

To address C1, we created a feature collection for fleeceware detection. This endeavor required us to draw heavily from the extensive knowledge presented in dark pattern literature, predominantly in the domains of human-computer interaction (HCI) and user experience (UX) design, as well as platform-specific requirements. We invited ten experts in Android and front-end development. They analyzed 1,486 user comments on fleeceware, identified seven common complaints, and executed 79 fleeceware samples, uncovering various phenomena in subscription UI. Finally, we extracted entities and attributes from the phenomena and identified 19 UI features for fleeceware detection.

C2: Extracting Features in the form of Natural Language. The most intricate aspect of the detection is extracting the values of the identified features from an app's UI. Most of our features are based on the information in the form of natural language, i.e., text describing a type of subscription information. However, the subscription information in a UI is usually fragmented and expressed in multiple formats due to the vast variability inherent in UI design, encompassing aspects like layouts, nested elements, and non-textual components. The extraction process requires a deep understanding of the relationship between the UI elements and the overall layout, as well as the capability to handle subscription information displayed in diverse formats.

To address C2, we developed a method that factors in both the UI elements and their layout information, incorporating multiple extraction rules. This method allows us to derive necessary information from the UI. Specifically, we developed a novel layout-based approach to link related subscription information by identifying *neighboring widgets*, which are descendant widgets of a parent widget like "ViewGroup" or "RelativeLayout" in the layout file. Additionally, we adopted a multi-rule-based approach to extract target information, avoiding potential semantic misunderstandings associated with machine learning. This approach is inspired by our analysis of 145 subscription UIs, where we observed developers consistently using specific keywords or symbols to describe subscription information, with numerical values consistently positioned within the information. We then constructed a collection of regular expressions to extract subscription details in various forms. This combined approach accurately extracts the target information for each UI feature.

With C1 and C2 addressed, we built a decision tree model for fleeceware detection. We chose this model because of its ability to incorporate our domain knowledge during feature engineering and its inherent model interpretability. To make it more intuitive for ordinary users to identify which interface features may present potential fleeceware risks, we employed SHAP, an Explainable Artificial Intelligence (XAI) technique, to provide and visualize the explanations.

Fleeceware Measurement. Owing to our automated detection capabilities, we are able to quantify the presence of fleeceware in the wild. We downloaded 13,597 apps from Google Play, spanning across all app categories, and identified 589 subscription-based apps. Within an approximate time frame of 10.12 minutes per app, we classified 443 of these apps, representing 75.21%, as suspected fleeceware, which may potentially cause unanticipated subscription costs for users. These apps have collectively garnered over 5 billion downloads. Even popular apps like *YouTube Music*, with over 1 billion downloads, contain fleeceware subscription UIs, indicating that the issues are significant. Our detection results are consistent with users' reviews on Google Play. With the detection results, we reported our findings through Google's app policy violation reporting platform to bring the problems to their attention.

Contributions. To sum up, the contributions are three-fold.

• We construct the first UI feature library for fleeceware detection, which is the result of collaboration with domain experts who possess in-depth knowledge of user interactions with fleeceware behaviors. They meticulously analyzed fleeceware samples and integrated insights from studies on dark patterns and platform-specific UI design requirements.

• We develop a novel technique that merges layout-based information linking with a multi-rule-based information extraction method for efficient subscription information extraction. This innovation stems from an extensive examination of subscription UIs, unveiling a consistent placement of relevant information within specific layout patterns and summarizing diverse presentation formats into common patterns.

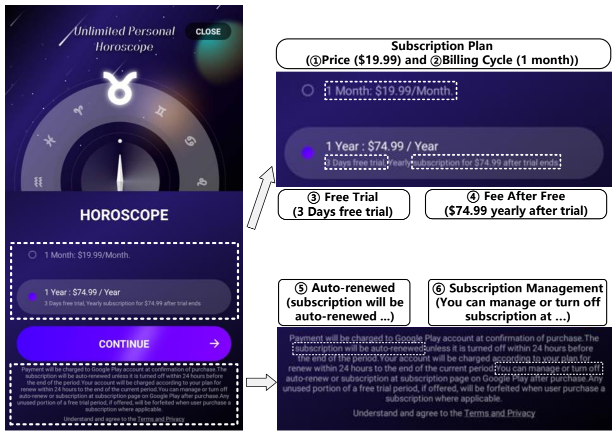
• We assess the prevalence of fleeceware throughout the app market and uncover its extensive distribution. Moreover, we investigate the developers, evolution, and app user perceptions of fleeceware, offering valuable insights to ethical developers, app users, and app market managers to mitigate its detrimental impact.

2 Background and Motivation

2.1 Subscription and Fleeceware

Subscription is a type of product offered in in-app purchase billing that allows app developers to sell content, services, or features through their apps and automatically charge users at regular intervals specified, e.g., a week or a month. To attract users, developers often set up a free trial period, allowing users to try out the subscription before purchasing. And users can cancel subscriptions before the billing cycle ends [27]. Figure 1 shows a subscription UI example, which typically displays information such as the subscription price, billing cycle, free trial details, etc.

Between 2015 and 2022, subscription revenue increased from around $1.5 billion to $17.1 billion [12]. As subscriptions gain popularity, they also become vulnerable to abuse


Unlimited Personal Horoscope
CLOSE
HOROSCOPE
1 Month: $19.99/Month.
1 Year : $74.99 / Year
3 Days free trial, Yearly subscription for $74.99 after trial ends
CONTINUE
Subscription Plan
(①Price ($19.99) and ②Billing Cycle (1 month))
③ Free Trial
(3 Days free trial)
④ Fee After Free
($74.99 yearly after trial)
⑤ Auto-renewed
(subscription will be auto-renewed ...)
⑥ Subscription Management
(You can manage or turn off subscription at ...)

Payment will be charged to Google Play account at confirmation of purchase.The subscription will be auto-renewed unless it is turned off within 24 hours before the end of the period.Your account will be charged according to your plan for renew within 24 hours to the end of the current period.You can manage or turn off auto-renew or subscription at subscription page on Google Play after purchase.Any unused portion of a free trial period, if offered, will be forfeited when user purchase a subscription where applicable.

Understand and agree to the Terms and Privacy

Figure 1: An example of a subscription interface

Table 1: Google requirements for subscription UI design

No.	Requirements for Preventing Subscription Abuse
R1	Be transparent about the offer, including the offer terms, the cost of the subscription, the frequency of the billing cycle, and whether a subscription is required to use the app.
R2	Users should not have to perform any additional action to review the subscription information.
R3	The content should accurately convey the meaning of the subscription. An example of a violation is “Free Trial” or “Try Premium membership - 3 days for free” for a subscription with an auto-recurring charge.
R4	Show how and when a free trial will convert to a paid subscription, how much the subscription will cost, and whether a user can cancel if they do not want to convert to a paid subscription.

by developers. **Fleeceware** [14] is a type of subscription app that deceives users into incurring unclear or hidden charges. Users may subscribe to apps without their consent, fail to realize they will be charged after a free trial, or struggle to cancel a subscription. Avast reported that the 204 fleeceware apps (70 in Android and 134 in iOS) they identified in 2021 have brought in 403.5 million dollars in revenue [4].

2.2 App Markets Regulatory Mechanisms

Ensuring effective oversight of apps on platforms like Google Play and AppStore poses significant challenges. The typical strategy is to employ a rating and review system to compile feedback from users. App reviews provide valuable information, including bug reports and feature requests [43]. App markets also use various vetting mechanisms, such as analyzing information uploaded by the app developer [49, 55], static scanning [60, 64], and dynamic execution [11, 25], to detect common malware. In addition, researchers have developed machine learning-based methods to detect new types of malware [17, 35, 41, 61]. However, the majority of these methods are ineffective at detecting fleeceware as fleeceware mainly utilizes hidden or contradictory content on the UI to confuse users, and exhibit coding patterns that closely resemble legitimate apps, without resorting to traditional malicious behaviors, such as stealing confidential data and causing system crashes.

Table 2: Dark patterns

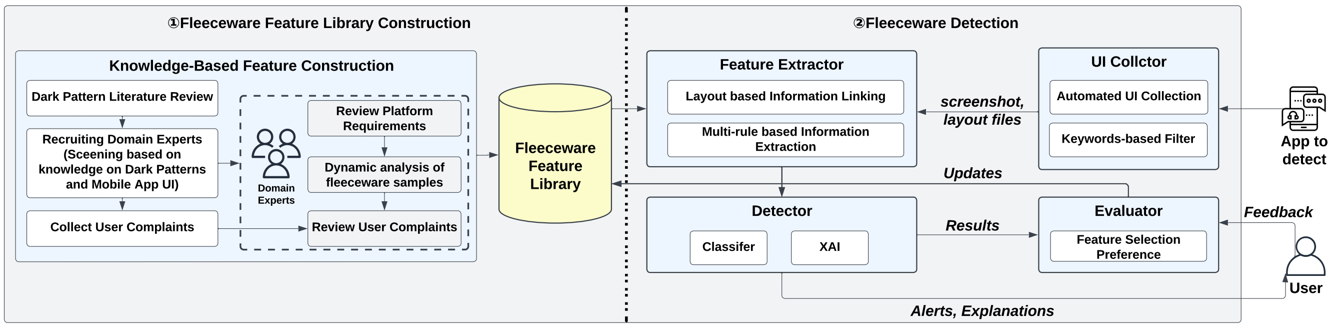
Category	Definition
Nagging	Redirection of expected functionality that persists beyond one or more interactions
Obstruction	Making a process more difficult than it needs to be, with the intent of dissuading certain action(s).
Sneaking	Attempting to hide, disguise, or delay the divulging of information that is relevant to the user.
Interface Interference	Manipulation of the user interface that privileges certain actions over others.
Forced Action	Requiring the user to perform a certain action to access (or continue to access) certain functionality.

Currently, app markets rely primarily on monetization requirements as the regulatory mechanism to combat fleeceware. For example, Google mandates that subscription app developers must not mislead users about subscription services or content offered within the app and provides specific guidelines for the development of subscription interfaces [28], as outlined in Table 1. However, relying solely on regulations is not sufficient to prevent the occurrence of fleeceware. In addition, due to the absence of automated detection methods, these markets are dependent on manual app reviews, a process that is neither timely nor scalable. It is crucial to identify the features which can indicate fleeceware, and build an effective tool for detecting them.

2.3 Dark Patterns Observed in Fleeceware

Fleeceware is a form of subscription fraud that is found to make use of dark patterns. To develop an automated fleeceware detection system, it is critical to employ the knowledge from dark pattern studies. Dark patterns have been extensively studied [10, 19, 21, 29, 40] in the human-computer interaction (HCI) and user experience (UX) fields. They are carefully crafted, deceptive interface design patterns that manipulate users into taking actions or making decisions they did not intend to make [10]. Gray et al. [29] have proposed a comprehensive categorization of dark patterns, including nagging, obstruction, sneaking, interface interferences, and forced action, as shown in Table 2.

In fleeceware, several dark patterns are observed. For example, the interface intentionally hides subscription-related information (Sneaking), causing users to unknowingly complete a subscription. The interface prominently highlights the “free trial” while making the subsequent payment information difficult to notice using small font sizes and inconspicuous colors (Interface Interference), which gives the impression that the app will not automatically charge fees. In sum, fleeceware leverages dark patterns to orchestrate subscription app scams, with most of these dark patterns observable in the subscription UI, leading to users’ financial detriment. This observation motivates us to detect fleeceware by identifying these dark patterns within the subscription UI.



①Fleeceware Feature Library Construction
②Fleeceware Detection
Knowledge-Based Feature Construction
Dark Pattern Literature Review
Recruiting Domain Experts (Sceening based on knowledge on Dark Patterns and Mobile App UI)
Collect User Complaints
Domain Experts
Review Platform Requirements
Dynamic analysis of fleeceware samples
Review User Complaints
Fleeceware Feature Library
Feature Extractor
Layout based Information Linking
Multi-rule based Information Extraction
screenshot, layout files
UI Collctor
Automated UI Collection
Keywords-based Filter
App to detect
Updates
Detector
Classifer
XAI
Results
Evaluator
Feature Selection Preference
Feedback
User
Alerts, Explanations

Figure 2: Framework of the fleeceware detection system.

3 DARKFLEECE

3.1 Threat Model

Our system is specifically designed to address a growing concern in the landscape of subscription apps: the use of deceptive UI designs (i.e., dark patterns) by app developers. These practices exploit gaps in user awareness and cognitive biases, leading to unwanted subscriptions and financial losses for users. To address these fleeceware problems, our work focuses on such deceptive design detection, and subscription UIs with such design are labeled as fleeceware UIs. Our aim is to highlight unethical design practices resembling financial fraud, contributing to a safer digital environment.

3.2 Overview

We introduce DARKFLEECE (Dark Pattern-based Fleeceware Detector), which methodically examines the UI of subscription apps to detect any existing dark patterns. The system framework is depicted in Figure 2. *Feature Library* is a key component, as the effectiveness of the system can greatly depend on its quality. This library is constructed by domain experts, drawing from observations of user interactions with fleeceware behaviors and a review of fleeceware samples, and integrates insights from dark pattern studies and platform-specific app requirements (address C1). DARKFLEECE then utilizes a *UI Collector* to collect subscription UIs from an app, and uses this feature library in its *Feature Extractor* to draw out pertinent feature values from the UIs. Within the extractor, we address the second challenge (C2) using a novel layout-based information linking technique and a multi-rule-based information extraction method to harvest target subscription information and obtain the feature values. Then a *Detector* module applies a trained classifier to detect potential fleeceware and utilizes an Explainable Artificial Intelligence (XAI) technique to provide explanations for the outcomes to make it easier for users to notice and understand the issues on the UI. As users provide feedback, an *Evaluator* module continually optimizes and updates the feature library to meet user needs. We will provide a detailed description of the specific design and implementation of each component below.

Table 3: Typical fleeceware behaviors users complain about

No.	Behaviors	Reviews from Google Play
B1	Claim to be free but actually not.	It says 3 days free trial, but when you press “continue” it makes you pay.
B2	Mislead users to subscribe without their knowledge.	I cannot believe I did not subscribe to this and they took £80.00 from my account.
B3	Make users confused about the charge plan.	Said it was $6.99/mo. Then it charged me $79.99 for a year.
B4	Do not state the recurring charge.	They billed me in June and again in July.
B5	Make users confused about canceling subscription.	I had uninstalled and canceled the subscription. I was still charged!
B6	Hard to close subscription UI.	The developer blended in the “X” in the corner so you can barely see it.
B7	The price is unreasonable.	I would never pay almost $100 A WEEK.

3.3 Knowledge-Based Feature Construction

Fleeceware identification necessitates a meticulous examination of fleeceware behaviors along with a solid understanding of dark patterns. Our first step involves gathering data on fleeceware behavior by accumulating user observations. Using these observations as a basis, we enlist domain experts to analyze user-observed behaviors. Additionally, experts analyze and execute a set of fleeceware to unearth any potential fleeceware patterns that may not have been observed by users. To accumulate user observations, we collected 1,486 pertinent comments on fleeceware reported by Avast [4] from Google Play. The complaints from users serve as a rich source of information for documenting fleeceware behaviors.

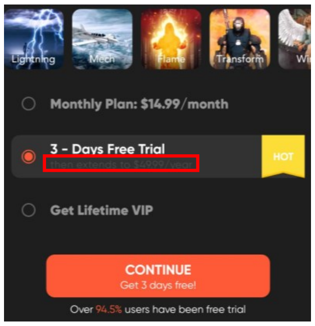
In the subsequent step, we engaged a panel of ten domain experts. This group was composed of two contributing authors of this study and eight additional volunteers, who were carefully selected through our professional networks to ensure a diverse yet highly specialized skill set was represented. The group consisted of four researchers and six engineers,

Table 4: The phenomena occurred in fleeceware subscription UIs

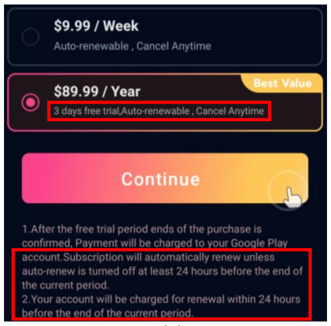
No.	Phenomena	Fleeceware Behaviors	Violation of Requirements	Category of Dark Patterns
1	The entire interface lacks price information.	B1, B2	R1, R2	sneaking
2	The entire interface lacks billing frequency information.	B1, B2, B3	R1, R2	sneaking
3	If there is a free trial, the entire interface does not inform about the trial period.	B1, B2	R2, R4	sneaking
4	The entire interface does not indicate whether the subscription will auto-renew.	B4	R1, R2, R4	sneaking
5	If there is a free trial, it does not specify the charges after the trial.	B1, B2	R4	sneaking
6	The total cost for a complete billing cycle is not provided.	B3	R1	sneaking
7	The total charge is not highlighted when multiple price formats are presented.	B3	R1	interface interference
8	A contradiction between the trial duration and the timing of the charge or cancellation of the subscription. (e.g., claim a 3-day free trial but charge on the last day of the trial.)	B2, B5	R4	interface interference
9	The interactive buttons do not convey the meaning of subscription (e.g., lack of keyword "subscribe" or not display the pricing information), and there is a significant distance between subscription information and the buttons.	B2	R1, R3	sneaking, interface interference
10	The subscription information is not prominent (e.g., small font size, similar color to the background, lack of emphasis in font style, buried within lengthy paragraphs).	B1, B2, B3	R1	sneaking, interface interference
11	The close button is not clearly visible.	B6	R1	obstruction, forced action
12	The subscription charges are unreasonable after the trial (e.g., charges after the trial significantly surpass market prices or prices with a free trial are higher than those without).	B7	-	interface interference

each bringing over four years of substantial practical experience in both Android and front-end design and development. These experts individually reviewed fleeceware samples alongside the documented fleeceware behaviors and met twice to discuss them. During the initial meeting, the experts summarized the documented fleeceware behaviors using the dark patterns in Table 3 to gain an understanding of the distribution of dark patterns in user observations. After the meeting, to discover potentially unobserved dark patterns, the experts further examined and ran 79 fleeceware samples, comprising 34 iOS apps and 45 Android apps. The results were discussed in the second meeting, and in cases of differing results, the experts engaged in discussions to reach an agreement. Remarkably, these fleeceware samples did not exhibit any notable anomalies or irregular behaviors during run time, with the exception of certain apps that frequently displayed advertising links. Nonetheless, the experts' analysis did uncover various dark patterns and breaches of platform requirements within these fleeceware subscription UIs. These findings, presented in Table 4, signify potential risk areas where users could inadvertently miss or misinterpret essential subscription information, mirroring problems outlined in Table 3.

Through expert analysis of the fleeceware samples, we have identified several crucial subscription details that may display characteristics of dark patterns. These details include the subscription cost, billing cycle, free trial period, and whether the subscription will automatically renew. It is imperative that these critical pieces of information be explicitly stated as required by the platforms (refer to the requirements in Table 1), as they play a crucial role in enabling users to make informed decisions regarding their subscription choices. The lack of these subscription details is identified in the subscription interface of the fleeceware samples (No.1-No.4 in Table 4).

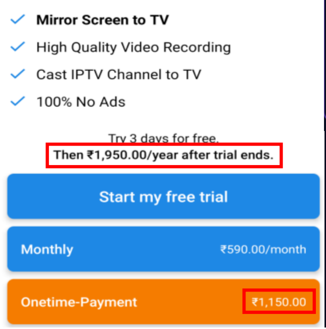

Lightning
Mech
Flame
Transform
Monthly Plan: $14.99/month
3 - Days Free Trial
then extends to $49.99/year
HOT
Get Lifetime VIP
CONTINUE
Get 3 days free!
Over 94.5% users have been free trial

(a)

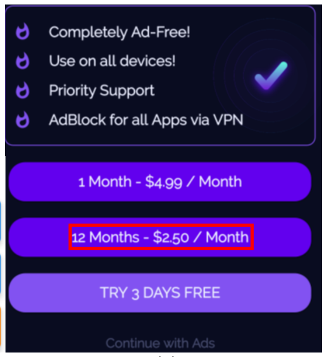

$9.99 / Week
Auto-renewable , Cancel Anytime
Best Value
$89.99 / Year
3 days free trial,Auto-renewable , Cancel Anytime
Continue

1.After the free trial period ends of the purchase is confirmed, Payment will be charged to your Google Play account.Subscription will automatically renew unless auto-renew is turned off at least 24 hours before the end of the current period.
2.Your account will be charged for renewal within 24 hours before the end of the current period.

(b)



(c)



(d)

Figure 3: Examples of fleeceware. 3(a) shows that the charge information is not easily discernible due to the small font size and color. 3(b) claims a 3-day trial, but users must cancel it 24 hours before the trial ends. 3(c) shows the price after the free trial is unreasonably higher than the price for a one-time payment, which reflects the characteristic of fleeceware, i.e., excessive charging. 3(d) displays a monthly price for an annual subscription, which violates Google's policies and may lead to user misunderstandings about the charges incurred.

Some fleeceware samples employ tactics to conceal or redirect users' attention away from critical subscription information (No.5-No.10). This can result in users inadvertently overlooking or misunderstanding the subscription details provided. Figure 3(d) showcases an interface that only presents a monthly price for a 12-month subscription (No.6), potentially misleading users regarding the long-term costs they will incur upon subscribing. Similarly, if a UI highlights the weekly price rather than the total price of the current subscription (No.7), it potentially causes users to underestimate the actual charges involved. Figure 3(b) claims a 3-day free trial, and users can cancel the subscription at any time. However, the fine print reveals that users must cancel one day before the trial ends to avoid automatic conversion to a paid plan and charges (No.8), which increases the risk of users missing the cancellation deadline and unintentionally incurring charges for the subscription. Interactive buttons are essential elements within a UI, serving as guides for user actions and triggering specific functionalities or navigation. Thus, the information displayed on these buttons significantly impacts users' understanding of their current actions. However, if an interface only mentions "Continue to trial" on the interactive button, while the information about automatically transitioning to an annual subscription after the trial period is displayed elsewhere (No.9). This design can mislead users into believing they are engaging in a trial period without realizing the subsequent charges. Font size, style, color, and positioning of informational text on an interface greatly influence users' ability to quickly and clearly perceive crucial information. In Figure 3(a), fleeceware employs small font sizes and colors that closely resemble the background, diverting users' attention from the billing information while emphasizing the "3-Days Free Trial" aspect (No.10).

Additionally, we found subscription interfaces that allow users to close the UI, implying that users can access the app without subscribing. However, in certain cases, the icon to close the UI is obscured or shares a color too similar to the background, making it difficult for users to notice (No.11), potentially leading them to mistakenly believe that subscription is the only way to continue using the app. Furthermore, some fleeceware samples demonstrate unreasonable pricing practices on their subscription interfaces (No.12). One such example is the app *Magic icon changer*, a wallpaper design app, which charges users a staggering $129.9 per week following a free trial period. Figure 3(c) illustrates an instance where the price after the free trial surpasses the cost of a one-time payment, further highlighting the exorbitant and unjustified pricing strategies employed by fleeceware apps.

Finally, we consolidate the observed phenomena identified by experts, extract the relevant entities and attributes, and formalize them into 19 features, as outlined in Table 6 in the Appendix C. Each feature serves as an indicator for specific aspects of the fleeceware subscription UIs. Specifically, Features F_1-F_5 and F_9-F_{10} are utilized to determine the presence of key subscription information for phenomena No.1-No.5 and No.9, respectively. Feature F_6 is derived from Phenomenon No.6 and signifies the observation that these interfaces often display more text describing the billing frequency than the actual price. Phenomenon No.7 is assessed using Feature F_7, which calculates the ratio of the font size of the total price to other prices, as well as examines whether the font is bold, indicating if the total price information is adequately highlighted. To determine the presence of Phenomenon No.8, we utilize F_8, which compares whether these two time are consistent. To evaluate the proximity between the subscription information and the interactive button in Phenomenon No.9, we employ Feature F_{11}, which measures the relative vertical location of these elements on the interface. For Phenomenon No.10, Feature F_{12} takes into account the relative length, relative font size, and font style of the price information compared to other text, assessing the visibility and noticeability of the information. Furthermore, Features F_{13}-F_{14} utilize optical character recognition (OCR) techniques to evaluate the clarity and perceptibility of the information presented on the interface. For Phenomenon No.11, Feature F_{17} employs edge detection techniques to determine whether an icon is clearly discernible. Lastly, we extract relevant price information and calculate Features F_{18}-F_{19} to represent Phenomenon No.12.

In total, we identified 19 features and will train a classification model based on them to detect fleeceware. In the next section, we will outline the process of extracting the values of these features from the UI.

3.4 UI Collector

The features we focus on are mainly related to UI entities and attributes that are displayed on the subscription UIs, e.g., the text contents and position of visual elements. These elements are static after the UI has been fully loaded and do not change dynamically during the user's interaction with the app. Therefore, we first need to obtain the fully-loaded subscription UIs. We utilize DroidBot [34], an automated tool capable of capturing screenshots and layout information of subscription UIs (including text, widget visibility, and layout details) at runtime, to extract UIs from apps.

However, it is not easy to capture subscription UIs within a limited time using DroidBot. This was due to the long search path with multiple branches, resulting in a timeout. To address this issue, we implement additional guided search strategies in DroidBot based on our prior experience. These strategies consider common patterns observed in subscription apps and adjust the priority of interactive events accordingly. *(1) Handling introductory screens:* In some apps, users are initially presented with a series of screens highlighting the app's features. To progress to the next page leading to the subscription UI, users need to click on buttons like "continue" or "start". We enable DroidBot to detect and interact with these buttons first. *(2) Scrolling down the interface:* During the introductory screens or other app UIs, users may need to scroll down to reveal the button or content necessary to proceed. To ensure that all contents are loaded before interacting, we prioritize the "ScrollEvent" in DroidBot's interactive events, allowing it to scroll down the page if necessary. *(3) Handling user input:* Some app UIs require users to fill in or select specific information before proceeding to the subscription UI. To handle this, we give higher priority to the "SetTextEvent" that fills text boxes in DroidBot, which ensures that DroidBot enters the required information before attempting to interact with buttons. Additionally, we pre-set fields such as nickname, birthday, email, age, etc., to facilitate input verification. And we use pre-configured Google accounts for requiring login. *(4) Prioritizing relevant keywords:* To effectively trigger the subscription UIs, we add relevant keywords to the "preferred lists" in DroidBot. These keywords include terms like "subscribe", "upgrade", "VIP", "premium". By prioritizing widgets associated with these keywords, DroidBot can efficiently reach the subscription UI. Finally, we utilize DroidBot on the crawled apps to generate screenshots and layout files for further analysis.

DroidBot saves the information of all the UIs encountered during the runtime. To improve efficiency, we use a keyword-based method to pre-filter UIs that are unrelated to subscriptions. Specifically, we collect all the texts in a UI, preprocess the texts by converting all words to lowercase, expanding abbreviations, removing non-ASCII characters and punctuation (keeping the currency symbols), removing stop words, and lemmatizing the words. Then, we filter out irrelevant UIs by checking for the presence of subscription-related keywords (e.g., "free", "trial", "$", "month", "subscription"), which are collected by counting the word frequencies in the subscription UIs previously gathered.

3.5 Feature Extractor

Each feature is composed of specific UI information, such as the presence of certain subscription details, text length, font size, etc. For example, to calculate F_{12}, we need to locate all subscription price information on the UI and extract the length or font size of the information and the whole sentence. However, accurately obtaining the information is difficult (challenge **C2**). The complexity primarily arises from two key aspects: (1) **Fragmentation of Information:** For example, in Figure 4, "Yearly" and "$29.99" are separated, and only when presented together as "Yearly $29.99", the meaning (i.e., subscription charge information) becomes clear to users. We need to design a method to effectively locate and concatenate these discrete pieces of information from the entire UI. (2) **Variable Display Formats:** For example, the charge information can be presented as "6 Months: $59.99" or "59.99 USD/6-Months". The diversity of information expression forms and the lack of dataset make it challenging for machine learning-based natural language processing methods to accurately differentiate between different subscription information and extract the values from them.

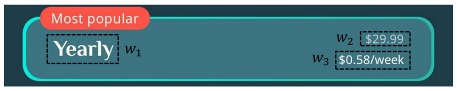


Figure 4: An example of *neighboring widgets*. w_1, w_2, w_3 are *neighboring widgets*, and they should be considered together.

Addressing the Fragmentation of Information: We design a novel layout-based approach to link related subscription information. Our idea is based on an observation that in a layout file, each piece of information is defined in a *widget* (a UI component that draws what users can see or interact with), and all the related information is defined in *neighboring widgets*. These widgets are descendants of a *parent widget* whose *widget class* is "ViewGroup", "RelativeLayout", "LinearLayout", etc. For example, in Figure 4, the three widgets containing charge information are *neighboring widgets*, which are children of a "ViewGroup" widget (i.e., the green box). Therefore, by searching for all the *neighboring widgets* of each widget, we can link all the related information.

Addressing Variable Display Formats: We adopt a multi-rule-based approach to extract target information, avoiding the potential semantic misunderstandings that could arise from machine learning approaches. This idea comes from the analysis of all the texts about subscription information from 145 subscription UIs obtained from the 79 fleeceware. We discover that the developer usually uses certain fixed keywords or symbols when describing each type of subscription information, and the numerical values appear in specific positions within the information. For example, the price and billing cycle information always appear together, within which, there will always be numerical values, currency symbols (e.g., "$", "USD"), and time units (e.g., "week", "month"). These elements appear in a specific order. Therefore, we construct a collection of regular expressions (available on our website[44]) to extract subscription details expressed in various forms. We detail how to generate the regular expressions and how to use the layout-based approach and the regular expressions to locate and extract target subscription information (including textual and visual information) as follows.

Layout-based textual information extraction. We first generate regular expressions using Regex Generator++ [6, 7] (an automated tool that creates text extraction patterns from given examples) as the initial reference. We manually fine-tune and expand these regular expressions based on specific cases encountered during the analysis. To extract target information, we need to combine all the related subscription information in a UI. For each widget, we find its closest *parent widget* with the target class, i.e., if a widget's parent w_p does not fall into the specified class, we continue searching for the parent of w_p. Once the *parent widget* is found, all the descendant widgets are considered as *neighboring widgets*. We link all the texts of *neighboring widgets* together and apply the regular expressions to get the target information.

However, directly applying the regular expressions to the combined texts may lead to many mismatches. This is because irrelevant information can add noise to the original texts and impact the accuracy of extraction. Moreover, when multiple useful pieces of information appear together, it may cause either information loss or incorrect information combination. To prevent mismatches, we consider performing real-time target information extraction during the process of linking the texts of *neighboring widgets*. Specifically, for a widget, if we can not extract any useful information from its text using the regular expressions, we search for its *neighboring widgets* and link their texts according to the layout, prioritizing widgets with smaller view IDs. The *view ID* uniquely identifies a widget and can be used to locate the widget. The widget with a small *view ID* comes first in a UI. We link the text in widgets in sequence, one by one. Assuming the previously concatenated text is t, then for a new *neighboring widget*, if its text does not contain any relevant information (i.e., numbers, words, and symbols mentioned in our regular expressions), we skip it. In contrast, if it matches our regular expressions, we extract the useful information, record its *view ID*, and link the remaining text to t. Once t matches the regular expressions, we extract the matched information and keep the remaining texts as t'. By doing so, we can accurately obtain all the necessary textual information and their location indicated by *view IDs*.

Visual information extraction. The proposed features include not only textual information but also visual information about the UI elements, such as font size, font style, and visibility. With the help of DroidBot, we managed to obtain information about widgets on the interface, such as the position of each widget containing subscription information according to the *boundary* field in the layout file indicated by *view IDs*, as well as the size of the widget view. However, we cannot directly obtain font-related information. To obtain the visual information of target information, we first retrieve the screenshots of related widgets and then utilize Tesseract-OCR [51], which can output the font size, style, and color of each word identified.

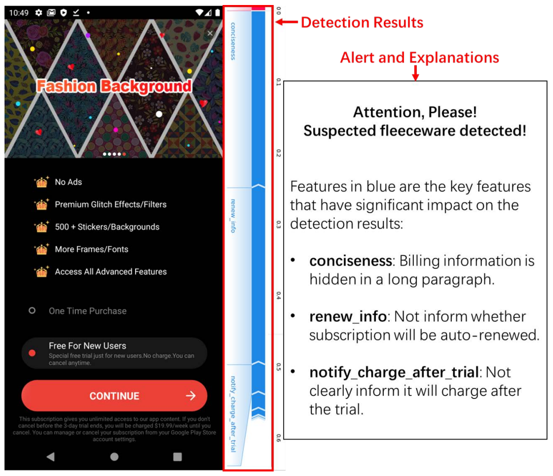
Since the text of a widget may contain more than just the target information, we need to match the target information with the strings recognized by OCR to obtain the desired visual information. Because OCR recognition can be inaccurate and does not always align with human perception, we optimize the matching process using the Levenshtein distance and Levenshtein ratio [63] to improve fault tolerance. The Levenshtein distance measures the minimum number of edit operations required to convert one string into the other, and the Levenshtein ratio is calculated by $\frac{sum-ldist}{sum}$, where sum is the total length of the two strings and $ldist$ is the Levenshtein distance between them. If the Levenshtein distance or Levenshtein ratio is below (or above) a predetermined threshold, we consider the strings to be a match and the target information to be clearly visible, and we can get the font size and style from the outputs of the OCR. If no strings match, we consider the information invisible.

For the visibility of a target icon, we apply the Canny Edge Detection technique [5, 22] to detect the icon in the designated area. If no edges are detected, the icon is considered invisible. Specifically, we identify the widget that represents the target icon by searching for keywords (e.g., “close”, “exit”) in the *resource id* field of each widget and get its screenshot. We then apply a multi-scale template matching method [31, 54] to recognize the icon. This method is based on the observation that target icons have several fixed styles, such as the shape of a cross for dismissing icons. The method is applied as follows: We first extract several icons with typical shapes as templates, and then slide the templates of different scales on the screenshots processed by the Canny operator and calculate the matching scores. Finally, we collect the icon detection results based on empirical thresholds for each template and vote for the final decision. If no templates match, the icon is regarded as invisible.

3.6 Detector

To account for which kind of UI may deceive users and make them overlook or misunderstand certain subscription information, a single feature may not be sufficient. We utilize machine learning techniques to determine the combinations of features and their respective weights that are indicative of problematic UIs. Considering the limited availability of training data and the fact that many features in our dataset are boolean values, we opted for a shallow Decision Tree Classifier model [9] to ensure accuracy while avoiding overfitting. To facilitate users' better understanding of detected issues on the UI, we need to provide explanations for the detection results. Although the Decision Tree inherently offers a certain level of interpretability, such interpretability might not be intuitive enough for ordinary users to understand the detection results, We integrated an additional step to address this issue. We employed SHAP (Shapley Additive exPlanations) [37, 38], an XAI technique, to visualize the most significant features contributing to the prediction results for each data sample. The integration of the XAI technique enables users to intuitively grasp the rationale behind individual classification results.

To create a reliable training dataset, we engaged ten experts, as detailed in Section 3.3, for three additional weeks to manually label the samples, dedicating around three hours each week to meticulous annotation. Additionally, these experts participated in an hour-long discussion weekly to ensure a consistent and accurate labeling process. The experts were presented with a set of 136 subscription UIs (crawled from the 45 Android fleeceware apps and another 45 subscription apps from Google Play), and their task was to independently assess whether each UI was a suspected fleeceware UI. If a UI was deemed to be suspected, the experts



10:49

Fashion Background

No Ads

Premium Glitch Effects/Filters

500 + Stickers/Backgrounds

More Frames/Fonts

Access All Advanced Features

One Time Purchase

Free For New Users

Special free trial just for new users.No charge.You can cancel anytime.

CONTINUE

This subscription gives you unlimited access to our app content. If you don't cancel before the 3-day trial ends, you will be charged $19.99/week until you cancel. You can manage or cancel your subscription from your Google Play Store account settings.

Attention, Please!
Suspected fleeceware detected!

Features in blue are the key features that have significant impact on the detection results:

- **conciseness**: Billing information is hidden in a long paragraph.
- **renew_info**: Not inform whether subscription will be auto-renewed.
- **notify_charge_after_trial**: Not clearly inform it will charge after the trial.

Figure 5: An example of the alert and explanation.

were instructed to mark the specific dark patterns that contributed to this assessment for the convenience of further discussion. An example of what experts need to label is shown in Figure 9 in the Appendix B. In cases where there were discrepancies, the experts engaged in peer discussions to reach a consensus. To this end, we got a collection of 76 suspected fleeceware UIs and 60 benign UIs. We then extracted UI features to form input feature vectors for training.

Using the labeled instances, we utilize default parameters provided by scikit-learn [45] to train a decision tree classifier. To help users understand the result, we utilize the SHAP technique, a game theoretic method for explaining machine learning outputs, to identify the primary features that contribute to the prediction outcome and visualize the explanations. Leveraging these insights, we offer alerts to users. The alert provided by DARKFLEECE for each identified suspected fleeceware UI highlights the specific elements or content on the UI that require attention. Figure 5 shows an example of the alert and an explanation of the identification. The visualization shows an area on the subscription UI where the user is only informed about the availability of a free trial, without being notified that the trial will automatically convert to a recurring charge after the trial period ends. Furthermore, the information about the fees is concealed in a lengthy paragraph. This alert assists users in circumventing the overlooking or misinterpretation of crucial subscription details.

3.7 Evaluator

Due to the lack of a clear definition for fleeceware and the variations in how different people perceive the severity of subscription issues, we aim to continuously adjust the set of subscription UI features based on diverse security concerns. On one hand, app markets and advanced users can run DarkFleece locally to detect fleeceware within apps. They can analyze their detection results according to their own security needs and adjust the required features accordingly, such as adjusting *PR_MAX* in F_{18}. On the other hand, DarkFleece can be run by third parties, where users upload apps and receive the results. Users can provide feedback, based on which, third parties can adjust the features most relevant and likely to cause issues for the users. With the support of the evaluator, DARKFLEECE can be personalized and continuously enhance its capabilities.

4 Evaluation and Findings

4.1 Setting

Between 2021 and 2023, we crawled real-world apps from Google Play and obtained 13,597 unique apps after removing duplicates based on their MD5 checksums. These apps span across all 33 categories on Google Play. To perform analysis on these apps, we utilized 10 workstations, each equipped with 4 cores with a 1.80 GHz CPU, 32.0GB memory, and 1TB hard drive. We set up a virtual machine on each workstation, with the virtual machine running Google Pixel4, featuring 2 cores, 2GB memory, and 16GB hard drives. The Android version used was 11.0.

4.2 Performance Evaluation

Effectiveness. We evaluate the effectiveness of DARKFLEECE based on the accuracy of extracting subscription UIs and identifying suspected fleeceware. For the first one, we randomly selected 100 apps that were successfully executed by DroidBot and collected 1,148 UIs, out of which 135 were subscription UIs after a manual check. Using our feature extractor, DARKFLEECE successfully extracted 140 subscription UIs, including all the 135 UIs above. Therefore, the accuracy of extracting subscription UIs is 99.57%.

For the second one, we re-selected UIs for evaluation considering the balance of the samples. Based on the results of DARKFLEECE, we selected 81 (10% of the total subscription UIs we collected) fleeceware UI samples and 81 benign samples. Considering the similarity of subscription UIs, we specifically sourced UIs from a diverse range of app categories to guarantee distinct UI designs. Additionally, we manually removed duplicate UIs to ensure an unbiased performance evaluation. We then asked the ten experts to label the samples and compared their results to assess the performance. Our analysis showed that 144 subscription interfaces are correctly labeled by DARKFLEECE, while 10 are mislabeled, including 7 false positives and 3 false negatives. Therefore, the accuracy of identifying fleeceware UIs is 93.83%. Overall, DARKFLEECE can achieve an accuracy of 93.43% (99.57% $\times$ 93.83%) for detecting fleeceware UIs.

We analyzed the reasons behind the errors in these two processes. The first reason is that developers don't follow traditional coding practice, such as not using the *Button* class to represent buttons, not using the *TextView* class to represent text, not using the *RelativeLayout* class to hierarchically structure the UI, as a result, the information we extract is inaccurate. The second reason is that there are advertising contents in the UI, which disrupts the target information.

Runtime Performance. On average, DARKFLEECE requires approximately 10.12 minutes per app for processing, which includes UI collection, UI pre-filtering, feature extraction, and fleeceware detection. Although collecting the UIs of an app could take up to 10 minutes, the subsequent procedures take less time, averaging 7.17 seconds per app. UI pre-filtering takes about 3.29 seconds per app (37,346.94 seconds for 13,597 apps), while feature extraction and fleeceware detection take an average of 3.88 seconds per app (4,373.03 seconds for 1,128 apps).

Users Study. To demonstrate the effectiveness of DARKFLEECE in helping app users pay attention to problems on subscription UIs and avoid mistakes, we conducted a survey study by recruiting participants from universities and collected 37 valid responses. Participants in the survey were presented with ten UIs from various subscription apps and asked to make an initial judgment regarding the UI's potential deceptiveness, without knowing the ground truth. Following their initial decisions, we asked them to review our detection results. The survey questions can be accessed on our website[44].

Specifically, we asked participants to identify suspicious UI issues and rate their confidence levels. And they were required to provide reasons for each selected UI. Subsequently, we provided them with explanations generated by DARKFLEECE, and asked them to re-select fleeceware UIs and rate their confidence levels again. 34 participants were undergraduate or graduate students, with one-third of them majoring in non-computer-related disciplines. The other 3 participants are family members, all aged 50 or above. Our study finds that 30 participants (81.08%) have encountered subscription problems before, and our explanations significantly improve users' ability to notice issues on fleeceware UIs. Specifically, our explanations on the 10 provided samples all draw more attention to the issues on the UIs, with an average increase of 5 participants noticing issues on the ten UIs provided. The most successful explanation resulted in 9 more (from 13 to 22) participants noticing the issues on the UI. All participants also reported higher confidence levels after receiving the explanations.

In addition, we observed that participants without a background in computer science and the three older individuals were more susceptible to fleeceware. Before we presented our results, these participants struggled to identify issues on the UI. Even when the UI employed only a few simple tricks, they were prone to overlooking crucial information. This was likely due to their limited familiarity with subscriptions and relatively less experience using mobile apps. This finding further supports that the features we proposed indeed contribute to potential harm. Moreover, through this user study, we discovered that the constructed features effectively covered the issues reported by participants, highlighting their comprehensiveness.

Assessing Flexibility in a Use Case. As outlined in Section 3.5, we recognize users' diverse perceptions of subscription issues and their differing tolerance levels towards fleeceware. Consequently, we equip users with the capability to modify the detection rate by customizing the features utilized in DARKFLEECE. We tested the flexibility by evaluating a use case. This use case addresses specific fleeceware concerns, particularly those involving the UI that emphasizes free usage while hiding pricing information, as well as cases where the claimed trial period is different from the actual billing time. With this specific concern, we re-evaluated and selected features from Table 6. We then re-annotated 100 subscription user interfaces based on these concerns. The features chosen were F_5 and F_8 through F_{13}. Out of these 100 samples, 67 were marked as "suspected", and 33 as "benign". DARKFLEECE accurately detected 93 of these samples, but misclassified 7, comprising 5 false positives and 2 false negatives. It shows that the accuracy rate for identifying fleeceware user interfaces remained at 93.00%, aligning closely with our earlier results.

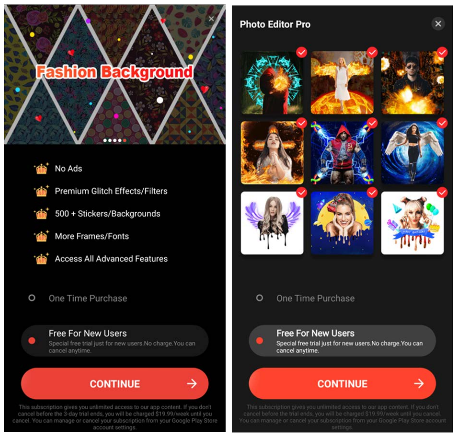
4.3 Findings in the Wild

Landscape. We analyzed 13,597 apps across all 33 Google Play categories, which were collected based on their popularity[1]. Surprisingly, out of the 589 apps with subscription UIs (813 in total), DARKFLEECE detected 443 (75.21%) apps as suspected fleeceware. These suspected apps have been downloaded over 5 billion times collectively. Moreover, each suspected fleeceware app had more than one fleeceware UI on average (629 UIs in 443 apps). Notably, some of the most popular apps, ranked in the top-200 list, were also identified as suspected fleeceware. In July 2022, we assessed the prevalence of fleeceware in popular apps ranked in the top 200 of the free, best-seller, and popularity lists. According to our findings, there were 7 suspected apps in the free list, 13 in the bestseller list, and 5 in the popularity list.

We also observed suspected fleeceware apps across all categories of apps. Table 5 in the Appendix A presents the distribution of suspected apps across various categories. Our investigation revealed that the issue was particularly widespread in the "Photography" and "Entertainment" categories, where the suspected fleeceware apps accounted for almost a quarter of all suspected cases. We report our findings to Google to get their recognition through their official feedback website[2].

[1]We used the website *https://app.diandian.com/rank/googleplay* for efficient categorizing and ranking.

[2]We reported through *https://support.google.com/googleplay/android-developer/contact/policy_violation_report*. Within the form, we categorized these apps under the "monetization and ads/subscription" category and specified their policy violation.



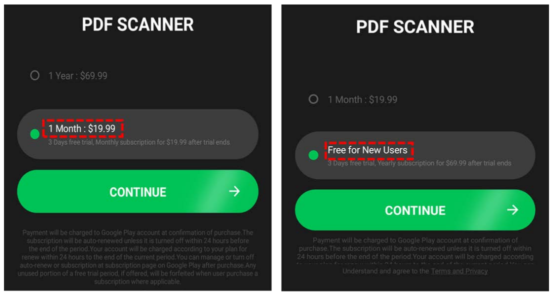
Special free trial just for new users.No charge.You can cancel anytime.

This subscription gives you unlimited access to our app content. If you don't cancel before the 3-day trial ends, you will be charged $19.99/week until you cancel. You can manage or cancel your subscription from your Google Play Store account settings.

Special free trial just for new users.No charge.You can cancel anytime.

This subscription gives you unlimited access to our app content. If you don't cancel before the trial ends, you will be charged $19.99/week until you cancel. You can manage or cancel your subscription from your Google Play Store account settings.

Figure 6: A case where two apps developed by the same developer were found to contain similar fleeceware UIs.



3 Days free trial, Monthly subscription for $19.99 after trial ends

Payment will be charged to Google Play account at confirmation of purchase.The subscription will be auto-renewed unless it is turned off within 24 hours before the end of the period.Your account will be charged according to your plan for renew within 24 hours to the end of the current period.You can manage or turn off auto-renew or subscription at subscription page on Google Play after purchase.Any unused portion of a free trial period, if offered, will be forfeited when user purchase a subscription where applicable.

3 Days free trial, Yearly subscription for $69.99 after trial ends

Payment will be charged to Google Play account at confirmation of purchase.The subscription will be auto-renewed unless it is turned off within 24 hours before the end of the period.Your account will be charged according

Understand and agree to the Terms and Privacy

Figure 7: A case that a UI becomes problematic from 2021 to 2022. The information "1 Month: $19.99" was changed to "Free for new Users". After the trial, it automatically switches to an annual subscription. But this information has a small font and a light color, making it easy for users to overlook.

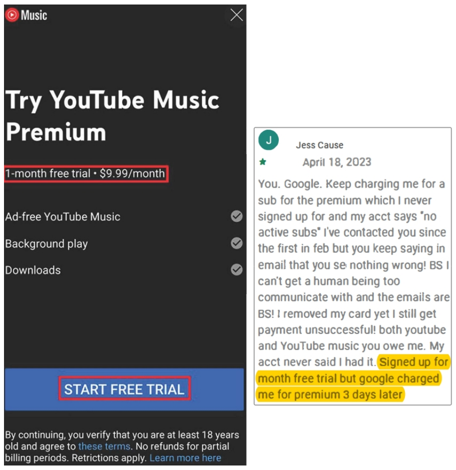
Common Strategies in Fleeceware UIs. Based on the interpretability outputs of DARKFLEECE, which indicate the features that contribute to UIs being classified as fleeceware UIs, we summarized the common strategies used in these UIs. The most common strategy observed is the lack of explicit indication on buttons about the ongoing subscription action (Phenomenon No.9, 270 in 629 suspected UIs, 42.93%), while emphasizing the free trial on the interface, e.g., not informing the user that the free trial will automatically transition into a paid subscription after the trial (Phenomenon No.5, 368 UIs, 58.51%), using a light font color and a small font size for the billing information (Phenomenon No.10, 182 UIs, 28.93%). In addition, 332 UIs (52.78%) do not clearly indicate whether the subscription will automatically renew (Phenomenon No.4). 111 UIs (17.65%) mislead users regarding the cancellation period (Phenomenon No.8). They claim to offer a 3-day free trial but require users to cancel one day in advance, otherwise, it will automatically convert to a paid subscription on the third day.

Fleeceware Developers. After collecting the names and contact details of developers of suspected fleeceware, we investigated them and found that apps developed by the same developer often have similar UIs with the same issues. 19 developers are found engaged in such practices. Figure 6 shows that the UIs of these two apps are almost the same and only emphasize the "free trial" option. Moreover, some developers are found to try to make more money by using different App IDs and names for nearly identical apps. For instance, apps named *Bravo cleaner* and *Bravo Security* are developed by the same developer, and have almost the same functionality and UIs. We also found that 12 suspected apps had no valid contact information or website for their developers. We suggest creating a blacklist for these developers and scrutinizing their apps more closely.

Fleeceware Evolution. We conducted a follow-up investigation to gain a better understanding of the evolution of suspected fleeceware over time. We selected 100 subscription apps (75 suspected and 25 benign) that were collected in June 2021. We downloaded their versions in August 2022 and their latest versions in June 2023. In 2022, 35 suspected fleeceware apps were taken down. Out of the 60 apps we were able to collect (39 suspected and 21 benign), only 2 suspected apps improved their subscription UIs and were classified as benign. However, we also identified 1 new suspected fleeceware app, as shown in Figure 7, where the price information of "1 Month: $19.99" was changed to "Free for new users", potentially misleading users into believing the app is free, and they may unknowingly be charged for a yearly subscription after the free period. In 2023, another 12 suspected fleeceware apps were taken down. Among the remaining 48 apps (24 suspected and 24 benign), 4 previously suspected apps became benign, but we also identified 1 new suspected app. Among the 4 apps transitioning to a benign model, we make a surprising discovery that an app called *PictureThis - Plant Identifier* adds a reminder feature, which notifies users before the trial period ends to remind them about the upcoming subscription fee. In summary, these findings demonstrate that app markets are making efforts to address the subscription issues, however, the continued emergence of suspected fleeceware suggests that developers are still motivated by the financial gains associated with fleeceware practices.

Generalizability for non-English apps. DARKFLEECE is built on UI features, allowing it to be applied to apps in different languages. However, adjusting keywords and regular expressions for specific languages is necessary when extracting feature values. On 25 subscription apps in Chinese and 25 in German crawled on September 6th, 2023, we identified 10 and 14 suspected fleeceware separately.

User Perceptions. User reviews are a valuable source for analyzing app issues, where we uncover numerous interesting subscription problems. Many users reported their children subscribing to an app without their consent or knowledge. Some users believed that the app would not automatically renew their subscription or deduct money if they did


Music

Try YouTube Music Premium

1-month free trial • $9.99/month

Ad-free YouTube Music

Background play

Downloads

START FREE TRIAL

By continuing, you verify that you are at least 18 years old and agree to these terms. No refunds for partial billing periods. Retrictions apply. Learn more here

Jess Cause

April 18, 2023

You. Google. Keep charging me for a sub for the premium which I never signed up for and my acct says "no active subs" I've contacted you since the first in feb but you keep saying in email that you se nothing wrong! BS I can't get a human being too communicate with and the emails are BS! I removed my card yet I still get payment unsuccessful! both youtube and YouTube music you owe me. My acct never said I had it. Signed up for month free trial but google charged me for premium 3 days later

Figure 8: The fleeceware UI detected in *YouTube Music*. The button only claims “FREE TRIAL”, and the UI states a 1-month free trial at the top, but users reported being charged 3 days later.

not have sufficient funds or uninstalled the app. Additionally, many users admitted to forgetting to cancel their subscriptions. These findings suggest that users are vulnerable to the threat of fleeceware due to their lack of awareness and caution when it comes to detecting and avoiding such scams.

We utilized user reviews as an indirect method of validating our experimental results. Specifically, we sampled the detected results and thoroughly examined all associated user reviews. Whenever we encountered user complaints about deceptive UI design within these reviews, we cross-referenced them with our detection results and explanations to determine whether our model had accounted for the issues highlighted by users. For example, the widely used application *YouTube Music*, which has been downloaded over 1 billion times, was detected fleeceware subscription UI as shown in Figure 8, and we have found user complaints related to the issue under the reviews of this application on Google Play (e.g., one said “Signed up for month free trial but google charged me for premium 3 days later”), indicating the potential risks. However, due to some newly published apps lacking reviews and many users not accurately or comprehensively expressing themselves when writing reviews, reviews cannot entirely reflect the issues existing in apps. Therefore, we primarily used reviews as a reference for validating the completeness of our features.

5 Lessons

The results of our investigation offer valuable lessons for app users, ethical developers, and app markets on how to avoid fleeceware issues and minimize their impact.

5.1 Ethical Developers

Some ethical developers may unintentionally introduce features that resemble fleeceware during development. They may not be aware that certain design choices could have an impact on users. Therefore, for these ethical developers, we provide some development recommendations to help them be more mindful. When designing subscription interfaces, developers should prioritize ethical practices by making user-friendliness a key consideration. This includes presenting subscription-related information clearly, accurately, and concisely in the UI. Important design considerations may include: (1) *Present subscription information in simple and concise sentences.* This may include stating the subscription plan and the cancellation deadline, such as “$9.99/week” and “cancel before the current period ends”. Providing a link or brief description of where users can unsubscribe (e.g., “Google Play - Settings - Subscriptions”) and clearly indicating if the subscription will automatically renew, using phrases like “auto-renewed” or “no automatic renewal”, can also be helpful. (2) *Highlight important information*, such as current charge information, with a larger font size to make it more visible. (3) *Place useful information near or on the interactive buttons*, like “subscribe” and “continue”, so that users can easily access the information before proceeding. (4) *Clearly display the dismiss icon in a familiar location*, such as the upper-left or upper-right corner, so that users can easily close the UI if they choose not to subscribe. (5) *Incorporate notification features to alert users of critical events*, such as subscription confirmations, cancellations, or approaching deadlines. These notifications can be in the form of pop-up messages or emails, and they can help users stay informed and avoid unexpected charges or renewals. Additionally, a simple test (such as a math question) before confirming a subscription can prevent accidental sign-ups by children without their guardians’ consent. Another useful suggestion is to use various widget classes in a standardized way when designing UI interfaces (especially in XML files). For example, use the “Button” class for buttons and the “TextView” class for text, which can greatly improve the efficiency and accuracy of software testing.

5.2 App Users

Lack of awareness and caution make many users vulnerable to the threat of fleeceware. To address this, we have developed a user manual to help users better understand subscriptions[44]. The manual covers crucial information such as what subscriptions are, what to pay attention to, how to cancel a subscription, and how to avoid common scams. For complete details, please visit our website. Some of the suggestions include: (1) *Be patient with subscription pop-ups*. Some apps will block users from accessing their services by displaying subscription pop-ups. If users really need the app, read the subscription terms carefully to avoid subscribing by mistake. (2) *Be cautious when navigating subscription UIs to avoid falling into traps*. Some apps deliberately hide subscription terms or make false claims about being free. Be careful before clicking any buttons in a UI, especially when seeing phrases like "use for free" or "free trial". (3) *Pay attention to essential subscription information*, such as the price, billing frequency, trial period, cancellation deadline, and auto-renewal terms when reading a subscription UI. Do not subscribe if this information cannot be found or understood easily. (4) *Understand the right way to cancel a subscription*. Uninstalling an app does not mean canceling the subscription. The common ways to cancel subscriptions are given in the user manual. Users can familiarize themselves with these cancellation ways before subscribing. (5) *provide feedback* to the app developers or app market managers as soon as possible if you encounter any problems.

5.3 App Markets

As a centralized platform for apps, the app market should strive to provide better services and enforce stricter management of the apps displayed on it to benefit users. (1) *A more user-friendly way to manage subscriptions* can be proposed to improve the user experience by providing. The information could be demonstrated on the download pages of all subscription apps to make it easy for users to find. (2) *Establish an effective feedback channel* for users to report issues about fleeceware and anything else. (3) *Using user reviews as a source for regulating apps.* In our research, we find that user reviews can reveal many issues with an app. These reviews not only help developers understand areas for improvement but also enable platforms to analyze and identify apps that pose potential threats. Platforms can then conduct further testing and monitoring of such apps to ensure compliance and user safety. (4) *Considering the use of a wider range of tools to detect potential risks in the app market.* The issues present in app marketplaces encompass various behaviors that can impact users. Market managers should consider employing different types of tools to discover and address these issues. Our tool can also help to detect suspected fleeceware in a cost-effective way. (5) *Consider implementing a reputation credit system* for developers and penalizing those who deliver fleeceware and other malicious apps.

6 Discussion

Deployment: DARKFLEECE currently operates as an offline-analysis tool that analyzes application packages and provides detection results before user interaction with the app, rather than during usage. Therefore, DARKFLEECE can be primarily used for offline analysis. App users can utilize it locally to detect fleeceware designs within apps before usage, averting potential deception. Benign app developers can use it to scan their apps before deployment to preemptively address any unintentional fleeceware issues. App markets can leverage it for large-scale scanning. Third-party app evaluators can also employ it to evaluate apps and share results with others.

DARKFLEECE can enhance detection performance and efficiency based on external information. For example, it can adjust detection strategies based on user feedback to achieve more customized and accurate detection. Furthermore, user reviews can raise alerts about potential fleeceware problems in the future, helping to refine detection targets and improve detection efficiency. For instance, when relevant user complaints are detected or formal reports are submitted, app markets can use DARKFLEECE to investigate potential fleeceware issues.

"Overcharge" Attribution Detection. Quantifying overcharge is subjective, any charge made without the end-user's knowledge constitutes an overcharge, so we emphasize identifying fleeceware's deceptive characteristics through dark patterns. In addition, we set *PR_MAX* in Feature F_{19} adjustable. End users can set this value based on their acceptable price range, and market regulators can set more appropriate values based on market statistics to achieve a more accurate estimation of the overcharge.

UI Static Analysis. The fleeceware features we focused on covered all types of dark patterns and platform requirements except for those related to dynamic interactions (i.e., the category *nagging* and requirement *R4*). Dynamic interactions were not considered for three reasons. Firstly, most subscription dark patterns and requirements for UI development are about static elements and layouts on the UIs. Secondly, the dark patterns related to interaction, such as subscription ads popping up repeatedly, may primarily affect the user experience when using the app rather than causing financial loss. Additionally, static analysis of subscription UIs can be more widely applicable as it can be conveniently used in any app, as long as the subscription UI can be obtained. However, subscription UI analysis may be affected by irrelevant elements on the UIs, such as in-app purchase information and pop-ups, which may cause DARKFLEECE to mistakenly analyze them as subscription UIs, resulting in false positives. We can further observe the features of these UIs and design methods to filter them out.

Features Extraction. DARKFLEECE extracts UI features by utilizing UI structure information, which may not be able to handle non-standardized development practices, such as the misuse of tags (e.g., misusing"TextView" and "Button"), displaying text within images, customizing view positions, complex interface hierarchies, and arbitrary naming of widgets. Further research could minimize dependence on the structure information by directly employing image processing techniques to extract and locate relevant elements within the subscription UI. This also enhances the robustness of our method. If attackers attempt to influence our extraction by altering the UI displayed to users, users may observe the issue and avoid being deceived.

In addition, regular expressions are utilized to extract important textual information. However, the vast number of apps in the market and the various expressions of information make it difficult to create regular expressions that cover all cases. To address this challenge and improve efficiency and comprehensiveness, an AI-based regular expression generator could be incorporated. Moreover, we used OCR techniques to extract visual information but encountered limitations that caused some elements to be inaccurately identified. Future research could leverage advanced computer vision techniques to obtain more precise information.

7 Related Work

Existing Android malware detection technologies mainly include static detection and dynamic detection. Static analysis performs feature extraction by disassembling source code to detect suspicious code without running the application [1, 2, 24, 32, 39], but cannot solve code obfuscation and dynamic code loading [3, 56]. Dynamic analysis can deal with code obfuscation by checking the characteristics of suspicious Android applications at runtime [13], but it will consume more resources and storage space [48], and malware can use anti-simulation methods to evade dynamic analysis [23, 57]. Recent deep-learning-based techniques can better identify unknown malware, but the effect depends on the algorithm design and dataset [59]. However, the existing techniques cannot detect fleeceware, because most fleeceware does not contain malicious code and exhibit coding patterns that closely resemble legitimate apps, without resorting to traditional malicious behaviors, such as stealing confidential data and causing system crashes.

Analyzing UIs can provide valuable insights for developers and designers to create effective UI design styles, accelerating the process of building UIs for their apps [8, 15, 18]. Moreover, UI analysis can reveal issues within an app, such as rendering problems and inefficient image display. DRAW [26] conducts UI rendering analysis to help developers identify and resolve short delays, while TAPIR [33] aims to detect inefficient image displays in mobile apps. Owl-Eye [36] can identify display issues, such as text overlap and missing images, and locate the specific region of the problem within a UI. Automated UI testing, a crucial aspect of UI analysis, involves dynamically exploring the user interfaces of an application to obtain relevant information. Various tools have been developed to facilitate this, such as Monkey [53], which generates a series of random operations (e.g., clicking, swiping) to execute an app automatically, and DroidBot [34], which uses a model-based strategy to enhance exploration efficiency. However, these studies and tools are mainly focused on functional testing and general display issues and are not adequate for detecting problematic subscription UIs. In our work, we utilize and modify DroidBot's search strategy to better detect subscription UIs.

Dark patterns are UI designs that deceive users into making decisions that do not align with their best interests. Recent research has focused on the negative impact of dark patterns and proposed solutions [10, 16, 19, 21, 29, 40, 42]. Gray et al. [29] categorize dark patterns into nagging, obstruction, sneaking, interface interferences, and forced action. Meanwhile, Narayanan et al. reviewed the history and ethical implications of dark patterns [42]. Additionally, Yada et al. [62] constructed a dataset for dark pattern detection, which comprised 1,818 dark pattern texts from shopping sites, and machine learning methods were applied to detect them. Furthermore, a study by Geronimo et al. [21] explored the prevalence and impact of dark patterns in mobile applications and suggested recommendations for designers and developers to create more ethical and user-friendly UIs that avoid the use of dark patterns. Chen et al. [16] uses general attributes of UI elements such as hierarchy, color, and font size to detect common UI dark patterns like layer overlays, pre-selected options, and hidden text. However, due to the unique deceptive practices of fleeceware, such as varying UI layouts and confusing textual expressions that convey ambiguous or contradictory semantics, this proposed approach lacks semantic analysis of interface information and is not effective for automatically detecting dark patterns on subscription UIs. We address this issue by extracting features related to specific subscription information and analyzing the overall accuracy and reasonableness of subscription information expression.

8 Conclusion

This work is dedicated to investigating fleeceware at scale through the development of an automatic fleeceware detection system called DARKFLEECE. To identify detective features for fleeceware, we construct a feature library based on expert knowledge drawing from observations of user interactions with fleeceware behaviors, a review of fleeceware samples, and insights from dark pattern studies and platform-specific UI design requirements. Then a novel layout-based information linking technique and a multi-rule-based information extraction method are designed to harvest subscription information, which is then converted to UI features. And a classifier is finally applied to detect whether the UI is suspected. With an accuracy of 93.43%, DARKFLEECE accurately identifies suspected fleeceware UIs and provides easily understandable alerts to users to comprehend the potential risks of fleeceware. We also ran DARKFLEECE in the wild to investigate the landscape, app developers, evolution, and user perception of fleeceware. Our findings offer valuable insights for app users, developers, and app market managers on how to prevent such problems. We have reported our findings to Google to get their recognition.

Acknowledgements

The IIE authors are supported in part by NSFC (92270204) and Youth Innovation Promotion Association CAS. And the HYU author was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant (No. RS-2022-00155885, Artificial Intelligence Convergence Innovation Human Resources Development (Hanyang University ERICA)) and the National Research Foundation of Korea (NRF) grant (NRF-2022R1F1A1074999), both funded by the Korea government (MSIT).

References

[1] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *Security and Privacy in Communication Networks: 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, September 25-28, 2013, Revised Selected Papers 9*, pages 86–103. Springer, 2013.

[2] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[3] Ömer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.

[4] Avast. Lists of fleeceware apps. `https://github.com/avast/ioc/tree/master/Fleeceware`, 2021.

[5] Paul Bao, Lei Zhang, and Xiaolin Wu. Canny edge detection enhancement by scale multiplication. *IEEE TPAMI*, 27(9):1485–1490, 2005.

[6] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[7] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Learning text patterns using separate-and-conquer genetic programming. In *EuroGP*, 2015.

[8] Farnaz Behrang, Steven P Reiss, and Alessandro Orso. Guifetch: supporting app design and development through gui search. In *MOBILESoft' 18*, pages 236–246, 2018.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees (cart). *Biometrics*, 40(3):358, 1984.

[10] Harry Brignull. Deceptive design. `https://www.deceptive.design/`, 2022.

[11] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *SPSM '11*, pages 15–26, 2011.

[12] BusinessofApps. Global consumer spending in subscription apps reached $17.1 billion in 2022. `https://www.businessofapps.com/data/app-revenues/`, 2022.

[13] Haipeng Cai, Na Meng, Barbara Ryder, and Daphne Yao. Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, 14(6):1455–1470, 2018.

[14] Jagadeesh Chandraiah. Fleeceware apps overcharge users for basic app functionality. `https://news.sophos.com/en-us/2019/09/25/fleeceware-apps-overcharge-users-for-basic-app-functionality/`, 2019.

[15] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th ICSE*, pages 665–676, 2018.

[16] Jieshan Chen, Jiamou Sun, Sidong Feng, Zhenchang Xing, Qinghua Lu, Xiwei Xu, and Chunyang Chen. Unveiling the tricks: Automated detection of dark patterns in mobile applications. In *Proceedings of the 36th UIST*, New York, NY, USA, 2023. Association for Computing Machinery.

[17] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the {Google-Play} scale. In *24th USENIX Security Symposium*, pages 659–674, 2015.

[18] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. Storydroid: Automated generation of storyboard for android apps. In *2019 IEEE/ACM 41st ICSE*, pages 596–607. IEEE, 2019.

[19] Gregory Conti and Edward Sobiesk. Malicious interface design: Exploiting the user. In *Proceedings of WWW '10*, page 271280, New York, NY, USA, 2010. Association for Computing Machinery.

[20] Alexandre Dewez. Benchmarking the pricing strategy of 100+ subscription based mobile apps. `https://alexandre.substack.com/p/-benchmarking-the-pricing-strategy`, 2020.

[21] Linda Di Geronimo, Larissa Braz, Enrico Fregnan, Fabio Palomba, and Alberto Bacchelli. Ui dark patterns and where to find them: A study on mobile applications and user perception. In *Proceedings of CHI '20*, page 114, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *Pattern recognition*, 34(3):721–725, 2001.

[23] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022, 2014.

[24] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security*, 65:121–134, 2017.

[25] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[26] Yi Gao, Yang Luo, Daqing Chen, Haocheng Huang, Wei Dong, Mingyuan Xia, Xue Liu, and Jiajun Bu. Every pixel counts: Fine-grained ui rendering analysis for mobile applications. In *IEEE INFOCOM 2017-ICCC*, pages 1–9, 2017.

[27] Google. Google play billing system overview. `https://developer.android.google.cn/google/play/billing`, 2022.

[28] Google-Developers. Policy center. `https://support.google.com/googleplay/android-developer/answer/9900533?hl=en&ref_topic=9857752`, 2022.

[29] Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. The dark (patterns) side of ux design. In *Proceedings of CHI '18*, page 114, New York, NY, USA, 2018. Association for Computing Machinery.

[30] ITRC. Subscription renewal scams are another way to steal your identity. `https://www.idtheftcenter.org/post/subscription-renewal-scams-are-another-way-to-steal-your-identity-itrc/`, 2021.

[31] Gábor Kertész, Sándor Szénási, and Zoltán Vámossy. Performance measurement of a general multi-scale template match-

ing method. In *2015 IEEE 19th INES*, pages 153–157, 2015.

[32] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *2015 IEEE/ACM 37th ICSE*, volume 1, pages 280–291, 2015.

[33] Wenjie Li, Yanyan Jiang, Chang Xu, Yepang Liu, Xiaoxing Ma, and Jian Lü. Characterizing and detecting inefficient image displaying issues in android apps. In *2019 IEEE 26th SANER*, pages 355–365. IEEE, 2019.

[34] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th ICSE-C*, pages 23–26, 2017.

[35] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. A review of android malware detection approaches based on machine learning. *IEEE Access*, 8:124579–124607, 2020.

[36] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. Owl eyes: Spotting ui display issues via visual understanding. In *35th IEEE/ACM ASE*, pages 398–409. IEEE, 2020.

[37] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839, 2020.

[38] Scott M Lundberg, Bala Nair, Monica S Vavilala, Mayumi Horibe, Michael J Eisses, Trevor Adams, David E Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2(10):749, 2018.

[39] Alejandro Martín, Héctor D Menéndez, and David Camacho. Mocdroid: multi-objective evolutionary classifier for android malware detection. *Soft Computing*, 21:7405–7415, 2017.

[40] Arunesh Mathur, Mihir Kshirsagar, and Jonathan Mayer. What makes a dark pattern... dark? design attributes, normative considerations, and measurement methods. In *Proceedings of CHI '21*, New York, NY, USA, 2021. Association for Computing Machinery.

[41] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. Deep android malware detection. In *Proceedings of the seventh ACM CODASPY*, pages 301–308, 2017.

[42] Arvind Narayanan, Arunesh Mathur, Marshini Chetty, and Mihir Kshirsagar. Dark patterns: Past, present, and future: The evolution of tricky user interfaces. *Queue*, 18(2):6792, may 2020.

[43] Ehsan Noei, Feng Zhang, and Ying Zou. Too many user-reviews! what should app developers look at first? *IEEE TSE*, 47(2):367–378, 2019.

[44] Our website. https://sites.google.com/view/study-about-subscription-uis/.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[46] Google Play. Subscription apps on google play: User insights to help developers win. https://services.google.com/fh/files/misc/subscription_apps_on_google_play.pdf, 2017.

[47] Thomas (TJ) Porter. What to do if youve become the victim of a subscription scam. https://www.mybanktracker.com/money-tips/money/subscription-scam-296704, 2022.

[48] Alireza Sadeghi, Hamid Bagheri, Joshua Garcia, and Sam Malek. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. *IEEE TSE*, 43(6):492–530, 2016.

[49] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Javier Nieves, Pablo G Bringas, and Gonzalo Álvarez Marañón. Mama: manifest analysis for malware detection in android. *Cybernetics and Systems*, 44(6-7):469–488, 2013.

[50] Andriy Slynchuk. How to know you're being scammed by a fleeceware app. https://clario.co/blog/how-to-spot-fleeceware-apps/, 2021.

[51] Ray Smith. An overview of the tesseract ocr engine. In *ICDAR 2007*, volume 2, pages 629–633. IEEE, 2007.

[52] Sophos. Dissecting fleeceware apps: the million-dollar money-making machine in android and ios. https://vb2020.vblocalhost.com/presentations/dissecting-fleeceware-apps-the-million-dollar-money-making-machine-in-android-and-ios/, 2020.

[53] Android Studio. Ui/application exerciser monkey. *developer. android. com. https://developer. android. com/studio/test/monkey (accessed Sep. 3, 2020)*, 2017.

[54] Feng Tang and Hai Tao. Fast multi-scale template matching using binary features. In *WACV '07*, pages 36–36. Citeseer, 2007.

[55] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker. Malware detection by applying knowledge discovery processes to application metadata on the android market (google play). *Security and communication networks*, 9(5):389–419, 2016.

[56] Sitalakshmi Venkatraman, Mamoun Alazab, and R Vinayakumar. A hybrid deep learning image-based analysis for effective malware detection. *Journal of Information Security and Applications*, 47:377–389, 2019.

[57] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ASIACCS*, pages 447–458, 2014.

[58] JAKUB VÁVRA. How fleeceware apps have earned over $400 million on android and ios. https://blog.avast.com/fleeceware-apps-on-mobile-app-stores-avast, 2021.

[59] Zhiqiang Wang, Qian Liu, and Yaping Chi. Review of android malware detection based on deep learning. *IEEE Access*, 8:181102–181126, 2020.

[60] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Seventh Asia joint conference on information security*, pages 62–69. IEEE, 2012.

[61] Ke Xu, Yingjiu Li, Robert H Deng, and Kai Chen. Deeprefiner: Multi-layer android malware detection system applying deep neural networks. In *EuroSP '18*, pages 473–487. IEEE, 2018.

[62] Yuki Yada, Jiaying Feng, Tsuneo Matsumoto, Nao Fukushima, Fuyuko Kido, and Hayato Yamana. Dark patterns in e-commerce: a dataset and its baseline evaluations. In *2022 IEEE International Conference on Big Data*, pages 3015–3022, 2022.

[63] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE TPAMI*, 29(6):1091–1095, 2007.

[64] Win Zaw Zarni Aung. Permission-based android malware detection. *International Journal of Scientific & Technology Research*, 2(3):228–234, 2013.

[65] Sergey Zubkov. Subscription prices have increased by 40%, whats next? https://adapty.io/blog/subscription-prices-have-increased-by-40-percent/, 2022.

Appendix

A Distribution of Fleeceware across Categories

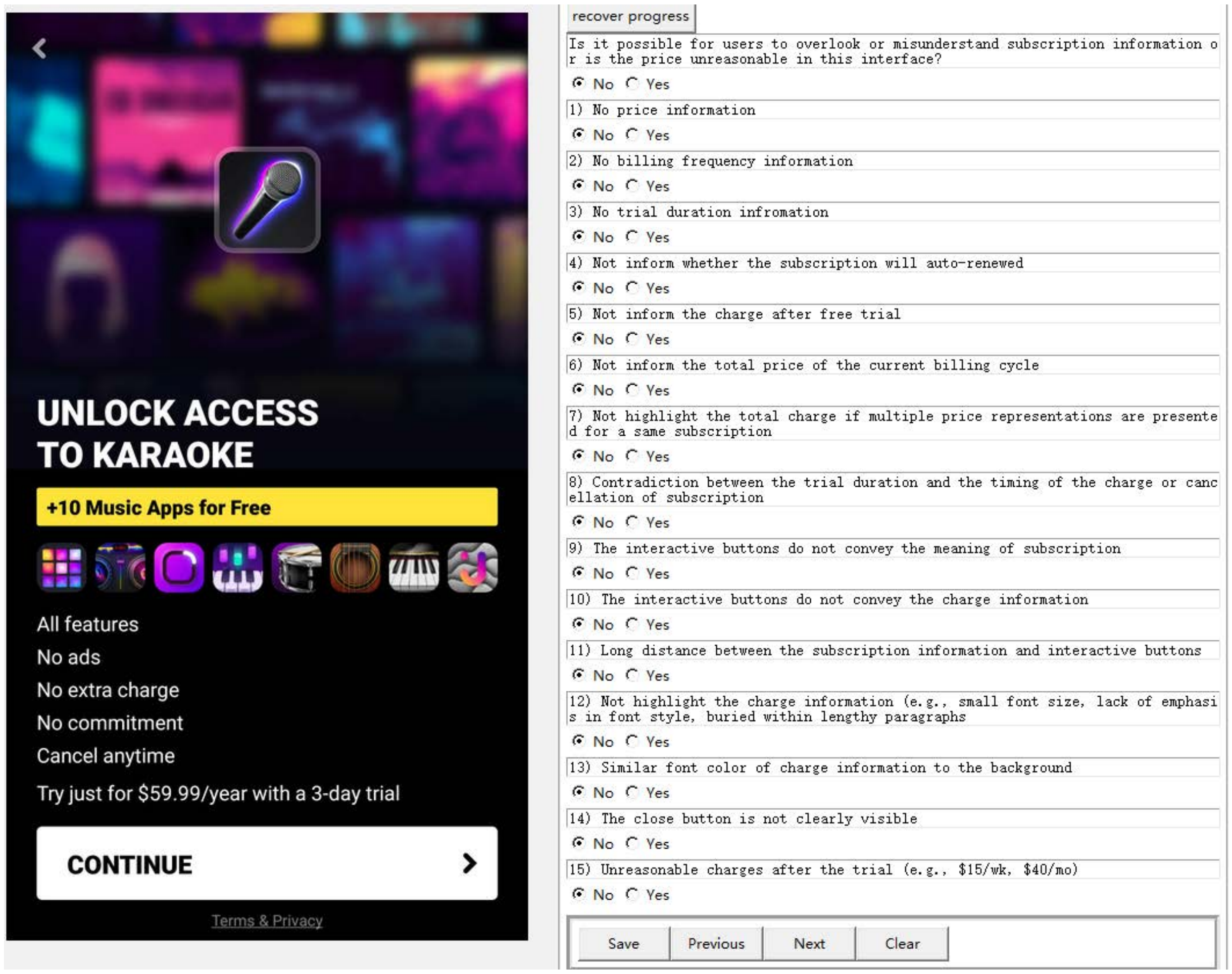
Table 5: categories of suspected fleeceware.

Categories	Number of apps	Ratio
Photography	48	10.84%
Entertainment	48	10.84%
Health & Fitness	28	6.32%
Music & Audio	27	6.09%
Personalization	25	5.64%
Tools	25	5.64%
Business	24	5.42%
Communication	21	4.74%
Video Players & Editors	20	4.51%
Education	18	4.06%
Lifestyle	18	4.06%
Weather	15	3.39%
Productivity	15	3.39%
Maps & Navigation	14	3.16%
Art & Design	12	2.71%
Others[1]	87	19.64%
Total	**443**	

[1] "Others" contains 18 other categories, e.g., "Comics", "Sports", "Events", "House & Home", "Travel & Local", each of which accounts for less than 2.5%.

B Label Tool

We developed an annotation tool to enhance the labeling efficiency of experts. The interface is shown in Figure 9.



UNLOCK ACCESS TO KARAOKE

+10 Music Apps for Free

All features
No ads
No extra charge
No commitment
Cancel anytime

Try just for $59.99/year with a 3-day trial

CONTINUE

Terms & Privacy

Is it possible for users to overlook or misunderstand subscription information or is the price unreasonable in this interface?
[✓ No] Yes

1) No price information
[✓ No] Yes

2) No billing frequency information
[✓ No] Yes

3) No trial duration infromation
[✓ No] Yes

4) Not inform whether the subscription will auto-renewed
[✓ No] Yes

5) Not inform the charge after free trial
[✓ No] Yes

6) Not inform the total price of the current billing cycle
[✓ No] Yes

7) Not highlight the total charge if multiple price representations are presented for a same subscription
[✓ No] Yes

8) Contradiction between the trial duration and the timing of the charge or cancellation of subscription
[✓ No] Yes

9) The interactive buttons do not convey the meaning of subscription
[✓ No] Yes

10) The interactive buttons do not convey the charge information
[✓ No] Yes

11) Long distance between the subscription information and interactive buttons
[✓ No] Yes

12) Not highlight the charge information (e.g., small font size, lack of emphasis in font style, buried within lengthy paragraphs
[✓ No] Yes

13) Similar font color of charge information to the background
[✓ No] Yes

14) The close button is not clearly visible
[✓ No] Yes

15) Unreasonable charges after the trial (e.g., $15/wk, $40/mo)
[✓ No] Yes

Figure 9: An example showing what users need to label.

C Features of Subscription UIs

Table 6: Features of Subscription UIs
(S is a given subscription UI)

Features	Phenomena
$F_1 = \begin{cases} 1 & \exists t \in T_S, T_{PI} \in t \\ 0 & \text{Otherwise} \end{cases}$, where T_S is the text set of the UI S, T_{PI} is the text of price information (PI).	No.1
$F_2 = \begin{cases} 1 & \exists t \in T_S, T_{BF} \in t \\ 0 & \text{Otherwise} \end{cases}$, where T_S is the text set of the UI S, T_{BF} is the text of billing frequency (BF) information.	No.2
$F_3 = \begin{cases} -1 & \forall t \in T_S, T_{FT} \notin t \\ 1 & \exists t \in T_s, T_{TD} \in t \\ 0 & \text{Otherwise} \end{cases}$, where T_{FT} is is the text of free trial information, T_{TD} is the text of trial duration information	No.3
$F_4 = \begin{cases} 1 & \exists t \in T_S, T_{AR} \in t \\ 0 & \text{Otherwise} \end{cases}$, where T_S is the text set of the UI S, T_{AR} is the text of auto-renewal information.	No.4
$F_5 = \begin{cases} 1 & \forall (t \in T_S \text{ and } T_{FT} \in t), T_{PI} \in t \\ 0 & \text{Otherwise} \end{cases}$, where T_{FT} is the text of free trial information, T_{PI} is the text of price information.	No.5
$F_6 = \begin{cases} 1 & \exists t \in T_S, N_{BF} \leq N_{PI} \\ 0 & \text{Otherwise} \end{cases}$, where N_{BF} and N_{PI} are the number of BF information and PI information in text t respectively.	No.6
$F_7 = \begin{cases} \min\limits_{t \in T_S, N_{PI} \geq 2 \text{ and } N_{BF} \geq 2} \left\{ k_1 \frac{fnt\_sz(T_{PI_1})}{fnt\_sz(T_{PI_2})} \times F_{bold}(T_{PI_1}) \right\} & \exists t \in T_S, N_{PI} \geq 2 \text{ and } N_{BF} \geq 2 \\ 2 & \text{Otherwise} \end{cases}$, where $fnt\_sz(T_{PI_2})$ and $fnt\_sz(T_{PI_2})$ are the font size of price information with the largest and the second largest billing frequency respectively. $F_{bold}(T_{PI}) = \begin{cases} k_2 & fnt\_sty(T_{PI}) = \text{“}bold\text{”} \\ 1 & \text{Otherwise} \end{cases}$, $fnt\_sty(T_{PI})$ is the font style of the text of PI, k_i is the weight parameters which can be adjusted.	No.7
$F_8 = \begin{cases} 0 & \exists T_{CD} \in T_S, val(T_{CD}) \neq val(T_{TD}) \\ 1 & \text{Otherwise} \end{cases}$, where T_{CD} is cancellation deadline for subscription, $val()$ outputs the value of an element.	No.8
$F_9 = \begin{cases} 1 & \forall b \in B_S, \text{“}subscri*\text{”} \in T_b \\ 0 & \text{Otherwise} \end{cases}$, where B_S is the button set in the UI S, T_b is the text of the button.	No.9
$F_{10} = \begin{cases} 1 & \forall b \in B_S, T_{PI} \in T_b \\ 0 & \text{Otherwise} \end{cases}$, where T_b is the text of interactive button in the UI S, T_{PI} is the text of price information.	No.9
$F_{11} = \frac{\min\limits_{T_{PI} \in T_S, b \in B_S} \{\text{dis}(ver\_loc(T_{PI}), ver\_loc(b))\}}{H_S}$, where $ver\_loc()$ outputs the vertical location of an element, H_S is the UI's height.	No.9
$F_{12} = \max\limits_{t \in T_S, T_{PI} \in t} \left\{ k_1 \frac{len(T_{PI})}{len(t)} \times k_2 \frac{fnt\_sz(T_{PI})}{fnt\_sz(t)} \times F_{bold}(T_{PI}) \right\}$, where $fnt\_sz()$ outputs the font size of the text.	No.10
$F_{13} = \begin{cases} -1 & F_1 = 0 \\ 0 & \exists T_{PI} \in T_S, obs(T_{PI}) = False \\ 1 & \text{Otherwise} \end{cases}$, where $obs(T_{PI})$ outputs whether the price information can be clearly observed.	No.10
$F_{14} = \begin{cases} -1 & F_2 = 0 \\ 0 & \exists T_{BF} \in T_S, obs(T_{BF}) = False \\ 1 & \text{Otherwise} \end{cases}$, where $obs(T_{BF})$ outputs whether the BF information can be clearly observed.	No.10
$F_{15} = \begin{cases} -1 & F_3 = -1 \text{ or } F_3 = 0 \\ 0 & \exists T_{TD} \in T_S, obs(T_{TD}) = False \\ 1 & \text{Otherwise} \end{cases}$, where $obs(T_{TD})$ outputs whether the trial duration information can be clearly observed.	No.10
$F_{16} = \begin{cases} -1 & F_4 = 0 \\ 0 & \exists T_{AR} \in T_S, obs(T_{AR}) = False \\ 1 & \text{Otherwise} \end{cases}$, where $obs(T_{AR})$ outputs whether the auto-renewal information can be clearly observed.	No.10
$F_{17} = \begin{cases} 1 & \forall ic \in IC_S, obs(ic) = True \\ 0 & \text{Otherwise} \end{cases}$, where IC_S is the icon set of a UI S, $obs(ic)$ indicates whether ic can be clearly observed.	No.11
$F_{18} = \begin{cases} 1 & \forall T_{PI} \in T_S, val(T_{PI}) \leq PR\_MAX \\ 0 & \text{Otherwise} \end{cases}$, where $val(T_{PI})$ is the value of PI, PR_MAX is the reasonable market price.	No.12
$F_{19} = \begin{cases} 1 & \forall\, T_{PI_1}, T_{PI_2} \in T_S, val(T_{PI_1}) \leq val(T_{PI_2}) \\ 0 & \text{Otherwise} \end{cases}$, where T_{PI_1} and T_{PI_2} are price with and without a trial.	No.12

Note: According to reports [20, 46, 65], we set PR_MAX as “\$15/week” and “\$40/month” in our work. The value can be changed according to user's expectation and the market price.