CHAMP: Characterizing Undesired App Behaviors from User Comments based on Market Policies

Yangyu Hu[1*], Haoyu Wang[2*✉], Tiantong Ji[3], Xusheng Xiao[3], Xiapu Luo[4], Peng Gao[5] and Yao Guo[6]
[1] Chongqing University of Posts and Telecommunications, Chongqing, China
[2] Beijing University of Posts and Telecommunications, Beijing, China
[3] Case Western Reserve University, USA [4] The Hong Kong Polytechnic University, Hong Kong, China
[5] University of California, Berkeley, USA [6] Peking University, Beijing, China

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) | 978-1-6654-0296-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE43902.2021.00089

***Abstract*—Millions of mobile apps have been available through various app markets. Although most app markets have enforced a number of automated or even manual mechanisms to vet each app before it is released to the market, thousands of low-quality apps still exist in different markets, some of which violate the explicitly specified market policies. In order to identify these violations accurately and timely, we resort to user comments, which can form an immediate feedback for app market maintainers, to identify undesired behaviors that violate market policies, including security-related user concerns. Specifically, we present the first large-scale study to detect and characterize the correlations between user comments and market policies. First, we propose CHAMP, an approach that adopts text mining and natural language processing (NLP) techniques to extract semantic rules through a semi-automated process, and classifies comments into 26 pre-defined types of undesired behaviors that violate market policies. Our evaluation on real-world user comments shows that it achieves both high precision and recall (> 0.9) in classifying comments for undesired behaviors. Then, we curate a large-scale comment dataset (over 3 million user comments) from apps in Google Play and 8 popular alternative Android app markets, and apply CHAMP to understand the characteristics of undesired behavior comments in the wild. The results confirm our speculation that user comments can be used to pinpoint suspicious apps that violate policies declared by app markets. The study also reveals that policy violations are widespread in many app markets despite their extensive vetting efforts. CHAMP can be a *whistle blower* that assigns policy-violation scores and identifies most informative comments for apps.**

***Index Terms*—User comment, app market, undesired behavior**

I. Introduction

Although the mobile app ecosystem has seen explosive growth in recent years, app quality remains a major issue across app markets [1], [2]. On the one hand, it is reported that millions of Android malicious apps were identified every year [3], using more and more complex and sophisticated malicious payloads and evasion techniques [4], [5], [6]. On the other hand, a large number of fraudulent and gray behaviors (e.g., ad fraud) were found in the mobile app ecosystem from time to time [7], [8], [9], [10], [11], [12]. Furthermore, apps with functionality/performance issues such as "*diehard apps*" [13], and devious contents such as "anti-society contents" still remain in the markets [14].

*The first two authors contributed equally to this work. Prof. Haoyu Wang is the corresponding author (haoyuwang@bupt.edu.cn).

Fig. 1. An example of user-perceived undesired behavior.

Most app markets have released strict developer policies, along with inspection and vetting processes before app publishing, seeking to nip the aforementioned threats in the bud and improve app quality in the markets. For example, Google Play has released a set of developer policies [15] that cover 10 main categories, including "Privacy, Security and Deception", "Spam and Minimum Functionality", and "Monetization and Ads", etc. Each category stands for a type of violation that may be associated with various undesired behaviors. Apps that break these policies should not be published on Google Play.

However, it is challenging to automatically check policy compliance for mobile apps. Despite Google Play's efforts in adopting strict vetting processes by using automated tools [16], [17], malware and Potentially Harmful Apps (PHAs) are recurrently found in Google Play [18]. Third-party app markets also show a significantly higher prevalence of malware, fake, and cloned apps [1]. On the one hand, it has been reported that many malicious apps use sophisticated techniques to evade automated detection [4]. For example, certain malicious behaviors could only be triggered at a specific time or environment, such as checking whether the app is being inspected in emulating environments [19]. On the other hand, even if malware can be detected by these automated tools, many other fraudulent and gray behaviors such as ad fraud and malicious push notifications are hard to identify. Moreover, functionality/performance issues are typically *app-specific*, while devious contents are broad and difficult to detect without human inspection, posing more challenges for automated tools [14].

In many cases, whether an app's behavior has exposed severe security risks or performance issues depends on how users think of it [20], [21]. As an important process for developers

978-1-6654-0296-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE43902.2021.00089

to improve app quality, app markets allow users to leave their ratings and comments after downloading and using each app [22]. These comments can be considered as the direct feedback from users who have experienced the apps [21], helping developers address the issues that might not have been spotted in testing. For example, as shown in Figure 1, two users gave 1-star ratings for the app. One user complained that this app contains aggressive advertising behaviors, and the other even reported that this app might be malicious. In fact, this behavior is also one of the *undesired behaviors* explicitly prohibited by the developer policies. When such comments are made aware to the market maintainers, they should be able to warn the app developers about the behaviors immediately and remove the apps from the market if such undesired behaviors are not addressed by the app developers. In other words, *user comments can form an immediate feedback for app market maintainers to identify user concerns and characterize the undesired behaviors that violate market policies.*

Ideally, user reviews could serve as an effective source for app markets to identify policy violations in the apps after they have passed the initial vetting process. However, the number of user comments in an app market is huge given the rapidly increasing number of apps, and there is a lack of automated tools to detect comments that are related to market policy violations and further perform deeper analysis on these comments. Furthermore, these useful comments are often buried in a much larger number of irrelevant comments, making it labor-intensive and error-prone to manually inspect these comments to obtain feedback. While user comments have been studied for emerging issues [23], app risks [20] and app recommendation [24], few research efforts have been spent in investigating how user comments can assist app markets in improving app vetting process. Thus, little is known *to what extent user comments can provide feedback on undesired behaviors that violate market policies and how app markets can utilize these feedback to improve their app vetting and maintenance process.*

In this work, we investigate the correlation between user comments and market policies, *i.e.,* characterizing user-perceived undesired behaviors prohibited by market policies. First, we create a taxonomy of 26 kinds of undesired behaviors summarized from the developer policies of 9 app markets. Then, we propose CHAMP, an approach that adopts text mining and NLP techniques to identify comments that describe these 26 kinds of undesired behaviors and classify them. We refer to such comments as ***undesired-behavior comments (UBComments)***. More specifically, CHAMP first extracts semantic rules from a training dataset of user comments via a semi-automated process. CHAMP then uses the extracted rules to automatically identify the undesired behaviors reflected in a given comment. Evaluation of CHAMP on benchmarks from real-word user comments suggests that it can successfully identify *UBComments* with high precision and recall (>0.9).

To further understand *UBComments* in the wild, we have curated a large-scale dataset from 9 app markets, with over 3 million user comments. We applied CHAMP on these comments to identify the *UBComments* and study their characteristics. We have a number of interesting findings:

- *UBComments* are prevalent in the app ecosystem, which can be found in 47% of the apps we studied. UBComments account for 20% for the 1-star comments. Our manual verification on sampled apps suggested the existence of undesired behaviors (96% of them could be verified). **It confirms our assumption that users can still perceive a large number of undesired behaviors prohibited by market policies, even though these apps have already passed the comprehensive vetting process**.
- User-perceived undesired behaviors, even some security-related ones, can be found in both malware and "benign" apps (the apps that were not flagged by any anti-virus engines on VirusTotal [25]). **It suggests that user comments can be a complementary source for providing insights of malware detection.**
- Although each market has explicitly declared developer policies, roughly 34% to 65% of apps in each market were still complained about their undesired behaviors against the policies. **This observation further indicates that it is hard for app markets to identify all policy violations during app vetting, while user comments could further help detect these violations continuously.** Moreover, policies from most markets are inadequate, as we have identified many apps (5% to 60%) showing undesired behaviors that are not covered in their policies.

To the best of our knowledge, this is the first large-scale study on the *correlation between user comments and market policies of mobile apps*. We believe that our research efforts can positively contribute to the app vetting process, promote best operational practices across app markets, and boost the focus on related topics for the research community and market maintainers. We have released the CHAMP tool, along with the policies and dataset to the research community at Github [26].

TABLE I
THE DISTRIBUTION OF POLICIES COLLECTED (TOTAL 599).

Market	# Policies	Market	# Policies
GooglePlay [27]	172	360 Market [28]	30
Huawei Market [29]	22	Lenovo Market [30]	28
Meizu Market [31]	53	Oppo Market [32]	15
Vivo Market [33]	96	Xiaomi Market [34]	159
Tencent Myapp [35]	24		

II. A TAXONOMY OF UNDESIRED BEHAVIORS

As we seek to identify the *UBComments* and investigate the correlation between user comments and market policies, we first collect a dataset of market policies and compile a taxonomy of the undesired behaviors described in them.

Market Policy Dataset. Considering that Google Play is the dominating market in the world except China, we seek to collect policies from 9 popular markets, including Google Play and 8 top Chinese third-party app markets, as shown in Table I. For each market, we crawl all the listed policies from

TABLE II
A TAXONOMY OF UNDESIRED BEHAVIORS AND THE DISTRIBUTION ACROSS MARKET POLICIES. THE ✓ REFERS TO THE MARKET DECLARING THE POLICIES. THE NUMBER REFERS TO THE # OF APPS WITH *UBcomments* WE IDENTIFIED FROM EACH MARKET IN SECTION VI.

Category	Behavior	360 Market	Huawei	Lenovo	Meizu	Oppo	Vivo	Xiaomi	Tencent Myapp	Google Play
Functionality and Performance	fail to install	264	✓(216)	✓(70)	✓(50)	33	✓(32)	✓(39)	✓(106)	✓(5)
	fail to retrieve content	30	33	5	✓(9)	10	✓(11)	✓(12)	✓(11)	✓(21)
	fail to uninstall	✓(119)	✓(46)	✓(11)	✓(19)	23	✓(29)	✓(21)	✓(49)	✓(1)
	fail to start (e.g., crash)	699	✓(451)	✓(209)	✓(238)	✓(174)	✓(318)	176	✓(880)	✓(105)
	bad performance (e.g., no responding)	334	✓(134)	30	60	✓(53)	✓(65)	✓(41)	176	✓(18)
	fail to login or register	180	201	✓(52)	88	98	✓(143)	86	184	✓(33)
	fail to exit	✓(62)	45	4	11	11	10	9	15	✓(2)
	powerboot	✓(3)	1	1	✓(0)	0	✓(0)	✓(3)	✓(0)	✓(5)
Advertisement	drive-by download	25	22	5	13	7	6	✓(14)	✓(9)	✓(25)
	ad disruption	✓(498)	262	✓(91)	✓(180)	✓(118)	✓(168)	✓(145)	✓(818)	✓(167)
	add shortcuts in launching menu	7	1	0	✓(7)	1	✓(0)	4	✓(4)	✓(7)
	ads in notification bar	15	1	✓(0)	✓(3)	1	✓(1)	✓(1)	✓(10)	✓(2)
Security	virus	✓(139)	✓(96)	✓(18)	✓(39)	✓(40)	✓(45)	✓(33)	✓(151)	✓(54)
	privacy leak	✓(25)	24	5	7	✓(9)	✓(16)	✓(11)	24	✓(30)
	payment deception	✓(236)	✓(189)	✓(39)	74	84	✓(127)	✓(61)	282	✓(75)
	illegal background behavior (e.g., sms)	160	109	24	**57**	51	✓(49)	✓(44)	✓(146)	✓(0)
	excessive network traffic	✓(90)	40	3	13	✓(25)	✓(30)	✓(16)	111	✓(4)
	hidden app	✓(12)	1	2	4	✓(0)	✓(0)	✓(1)	2	✓(1)
	illegal redirection	80	35	✓(5)	17	20	✓(19)	✓(16)	135	✓(8)
	permission abuse	37	✓(27)	4	✓(8)	4	✓(4)	✓(17)	✓(11)	✓(27)
	illegitimate update (e.g., update to other app)	3	3	✓(0)	0	3	1	2	1	✓(0)
	browser setting alteration	0	0	0	0	0	✓(0)	✓(0)	0	✓(0)
Illegitimate Behavior of Developers	app repackaging	132	16	12	✓(11)	14	17	✓(13)	64	✓(14)
	app ranking fraud	54	28	7	✓(34)	22	✓(20)	✓(21)	45	✓(6)
Content	vulgar content (e.g., pornography, anti-society)	✓(47)	18	✓(1)	✓(6)	4	✓(8)	14	✓(21)	✓(15)
	inconsistency between functionality and description	15	5	✓(0)	2	3	8	✓(1)	8	✓(1)
Total # of apps with undesired behaviors		1025	625	338	422	237	463	257	1382	274
Total # of apps with undesired behaviors (declared policies)		731	537	318	365	210	460	211	1233	274
Total # of apps with undesired behaviors (undeclared policies)		891	433	90	219	178	36	191	654	0

the corresponding webpages. In total, we have collected 599 policies. Note that the developer policies of Google Play were in English, while the other market policies were in Chinese. Google Play has more complete and fine-grained policies than any of the third-party app markets.

Summary of Undesired Behaviors. As the policies defined by each market vary greatly (some are coarse-grained and some are fine-grained), it is non-trivial to automatically classify them. Thus, the first two authors of this paper manually went through these policies, and classified them into 5 main categories, including 26 distinct undesired behaviors. Table II shows the taxonomy of the summarized undesired behaviors, and the distribution of the corresponding policies across markets. Note that one behavior may correspond to one or more market policies. We observe that all of the undesired behavior regulations can be found in Google Play. As for the third-party markets, `Vivo` and `Xiaomi` have declared policies related to the most types of undesired behaviors, covering 21 and 20 behaviors respectively. We believe that this taxonomy covers most of the commonly observed undesired behaviors. Even though it may still be incomplete, our approach is generic and can be adapted to support new behaviors and different granularities of behaviors (see Section VII).

III. AUTOMATED CLASSIFICATION OF UBCOMMENTS

A. Overview

Figure 2 shows the overview of CHAMP, which builds a training dataset of user comments (the *training dataset building phase*), extracts semantic rules from the labelled comments (the *semantic rule extraction phase*) and uses the rules to identify and classify *UBComments* (the *detection phase*). The major reason why we prefer semantic rules instead of text similarity is that most comments are short and often use a few key phrases in specific orders such as "icon disappears", while semantic rules have shown promising results in identifying sentences with specific purposes [36], [37], [38]. On the contrary, text similarity approaches based on word similarity without emphasis on key phrases are optimized for general purposes, and thus these approaches require extra tuning to focus on certain words that play important roles in the sentences of market policies [39], [40], [41]. Additionally, these approaches generally require a substantial amount of labelled samples to train the weights, which is less effective in our context due to the limited number of labelled samples.

❶ In the training dataset labelling phase, we collect the comments of the apps from Google Play and 8 third-party app markets, and resort to text clustering model to help to label the user comments. In the topic modeling and topic labelling

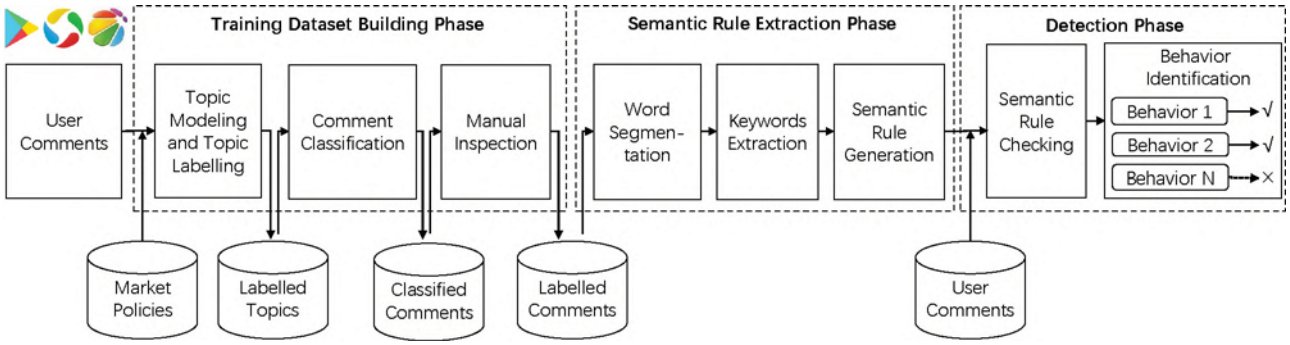

Fig. 2. Overview of CHAMP.

step, we first merge the market policies that describe a same undesired behavior into a single document (26 documents in total). Then, CHAMP applies a short-text topic modeling algorithm [42], [43], [44], [45] to identify a set of topics, where each topic contains a set of words. At last, CHAMP labels each topic with related undesired behavior based on the similarity between the documents of policies and the words in the topics. In the comment classification step, CHAMP uses the labelled topics to classify each comment into related undesired behaviors. We further manually inspect the classified comments to confirm whether these comments are related to the corresponding undesired behavior. This is necessary because we could only classify each comment based on the keywords with the highest weight under each topic, which may introduce false positives. For example, if a comment contains the keyword "notification", it is considered to be likely to related to the behavior "ads in notification bar". However, the word "notification" may also appear in comments that talk about alerts and notifications (e.g., notifications and alerts for weather apps). ❷ In the semantic rule extraction phase, CHAMP applies a generation algorithm on the labelled comments and generates semantic rules for each undesired behavior automatically.❸ In the detection phase, CHAMP accepts user comments as input, and uses the semantic rules to classify comments into the undesired behaviors defined in market policies.

B. Training Dataset Labelling

Training Dataset. To label training dataset, we randomly select 2% of the comments for each app in our dataset (discussed in § IV). In total, we extract 70,000 comments, including 15,000 English comments and 55,000 Chinese comments. Note that these comments were used separately for training two models for both English and Chinese comments.

Topic Modeling and Topic Labelling. Unlike traditional documents (*e.g.,* news articles), the descriptions of undesired behaviors in market policies consist of only one or a few short sentences. Thus, the lack of rich context makes it infeasible to use the topic modeling algorithms such as PLSA [46] and LDA [47], which implicitly model document-level word co-occurrence patterns. To address this problem, we apply BTM (biterm topic model) [42], a widely used model for short-text topic modeling, to learn the set of topics for market policies. BTM explicitly models word co-occurrence patterns using *biterms*, where each biterm is an unordered word-pair co-occurred in a short context. The output of BTM are a set of topics where each topic consists of a list of words and their weights. For each topic z, BTM draws a topic-specific word distribution $\phi_z \sim Dir(\beta)$, and draws a topic distribution $\theta \sim Dir(\alpha)$ for all of the documents, where α and β are the Dirichlet priors. For each biterm b in the biterm set B, it draws a topic assignment $Z \sim Multi(\theta)$ and draws two words $(w_i, w_j) \sim Multi(\phi_z)$, where w_i and w_j are words appearing in the same document. Following the above procedure, the joint probability of a biterm $b = (w_i, w_j)$ can be written as:

$$P(b) = \sum_z P(z)P(w_i|Z)P(w_j|Z)$$

thus the likelihood of all the documents is:

$$P(B) = \prod_{(i,j)} \sum_z \theta(z)\phi_{i|z}\phi_{j|z}$$

We conduct topic modeling based on the merged English and Chinese policies, respectively. We set the number of topics as 26, which corresponds to the number of undesired behaviors. CHAMP then labels the proper undesired behaviors for the topics by computing the probability of each document being allocated to each topic. It assumes that the topic proportion of a document equals to the expectation of the topic proportion of generated biterms during topic modelling:

$$P(z|d) = \sum_b P(z|b)P(b|d),$$

where z represents topic, b represents biterm and d represents document. $p(z|b)$ can be calculated via Bayes formula based on the parameters estimated in BTM:

$$P(z|b) = \frac{\theta_z \phi_{i|z}\phi_{j|z}}{\sum_z \theta_z \phi_{i|z}\phi_{j|z}}$$

TABLE III
REPRESENTATIVE STOPWORDS USED IN CHAMP.

Removed Stopwords	Added stopwords
miss, high, ask, give, can not, how, able, stop, without, allow, obtain, other	god, sex, s**t, s**d, silly, blah, r**h, d**n, d**b, da*n, horrible

$p(b|d)$ can be estimated by the empirical distribution of biterms in the document, where $n_d(b)$ is the frequency of the biterm b in the document d:

$$p(b|d) = \frac{n_d(b)}{\sum_b n_d(b)}$$

At last, CHAMP selects the highest score of P(zld) and labels the proper undesired behaviors for each of the 26 topics.

Comment Classification. CHAMP then classifies each comment into related topics. It computes the probability of each comment being allocated to each topic. If the probability is above a certain threshold, CHAMP considers that the comment is related to the topic. We follow the same empirical approach [43], [44], [45] to set the threshold, and find that 0.6 is a good indicator. In total, we obtain 9,228 comments that are related to 26 distinct behaviors.

Manual Inspection. Considering that the automated classified comments may be not related to the undesired behaviors (see § III.A), we further manually inspected the comments that are classified into related topics to confirm whether these comments are *UBComments*. Besides, if a comment is related to more than one behavior, we split the comment into several sentences and each sentence is related to a kind of undesired behavior. Two authors inspect the comments independently. For the disagreements of category labelling, a further discussion is performed. Eventually, we obtained 8,275 comments that are related to 25 distinct behaviors. After splitting some comments, we obtained 9,057 labelled comments in total, which will be used for semantic rules generation. Note that we did not find any comments that are related to the behavior of "browser setting alteration".

C. Automated Semantic Rule Extraction

Based on the labelled comments, given a new comment, the goal of CHAMP is to determine whether the comment describes the same or similar behavior as the labelled comments. To achieve this goal, we propose to automatically extract *semantic rules* from the labelled comments for each undesired behavior. Firstly, for each undesired behavior, CHAMP extracts and sorts the representative words from the related comments. Then, CHAMP analyzes the relations of the keywords by merging the keywords that usually appear in the same comments. After that, we can get one or more keyword sets containing different representative keywords. At last, CHAMP generates semantic rules for each keyword set by combining keywords and calculating the distance constraints of the keywords.

Word Segmentation. In this step, CHAMP groups the comments related to each undesired behavior into a corpus (25 corpora in total). For each corpus, it segments the comments into words, removes meaningless words and sorts the remaining words in descending order based on the TF-IDF [48] weighting to generate a word list $WordList$. Stopwords are the words considered unimportant in text analysis tasks. Thus, we take advantage of the stopword lists provided by HIT [49] and a public English stopwords list "stopwords-iso" [50]. *However, we find that the general stopword lists cannot well fit the app comment study*. On one hand, when some traditional stopwords (*e.g.,* can) are combined with other words, they become key phrases for describing undesired behaviors in user comments. For example, the comment "always have to download other apps" is related to the undesired behavior "drive-by download", thus the traditional stopwords "always" and "other" should not be removed. We summarized and removed 29 stopwords (including 14 English stopwords and 15 Chinese stopwords) that are important for describing undesired behaviors from the stopwords list. On the other hand, existing research found that there exist noises and spams (e.g., offensive comments) in app comments, which are meaningless for describing undesired behaviors. Therefore, we adapt the selected stopword list and add over 50 new stopwords that are regularly appeared in user comments. The representative stopwords are shown in Table III (offensive words are sanitized).

Representative Keywords Extraction. The goal of this step is to identify the most representative keywords that can cover the labelled comments in a given corpus. Thus, for each keyword in the *WordList* of a given corpus, CHAMP first collects the comments in the corpus that contain the keyword and adds them into a comment set $ComtSet_{word}$. Then, a traversal operation begins to select the keywords in order (based on TF-IDF weight) and compare the $ComtSet_{word}$ of different words. For the comment set $ComtSet_{word_m}$ of the m-th word $word_m$ in the $WordList$, if part of the comments in it are overlapped with the comments in the n-th word's ($n < m$) comment set $ComtSet_{word_n}$, CHAMP will merge $word_m$ and $word_n$ into a keyword set. Otherwise, CHAMP will assign the word $word_m$ into a new keyword set. Note that, the traversal operation will stop if the union set from $ComtSet_{word_1}$ to $ComtSet_{word_m}$ contains all of the labelled comments in the corpus. Based on the traversal operation, CHAMP could extract one or more keyword sets for each corpus.

Semantic Rule Generation. For each of the extracted keyword sets in a corpus, CHAMP automatically generates semantic rules. We observe that a behavior can be generally described by two keywords of different part-of-speech [51] in a comment. For example, the verb "steal" and the noun "money" in the comment "it steals money from the credit card!!!" are related to behavior "payment deception". Another example, the adverb "how" and the verb "uninstall" in the comment "who can tell me how to uninstall this app" are related to behavior "fail to uninstall". Thus, for the extracted keyword sets, CHAMP combines the keywords of different part-of-speech pairwise. Furthermore, we observe that most

TABLE IV
REPRESENTATIVE SEMANTIC RULES FOR 4 BEHAVIORS.

Behavior	semantic rules
virus	{virus, null, null} {trojan, null, null} {malware, null, null}
ads in notification bar	{notification, ads, 3} {notification, full, 2} {remove, notification, 4}
permission abuse	{ask, permission, 5} {require, permission, 6} {unnecessary, permission, 2} {need, permission, 6} {want, permission, 7}

UBComments are short and often include key phrases in specific orders. Therefore, the semantic rules not only contain keywords but include order and distance constraints on matching the keywords. For two keywords $keyword_u$ and $keyword_v$ ($ComtSet_u \cap ComtSet_u \neq \emptyset$), CHAMP will generate two semantic rules $\{keyword_u, keyword_v, constraints\}$ and $\{keyword_v, keyword_u, constraints\}$, the constraints is used to limit the distances of these two keywords. For example, semantic rule $\{ask, permission, 3\}$ means that "ask" appears before "permission" and their distance is less than 3 words. CHAMP automatically calculates the F1-score under different distance constraints (we set it range from 1 to 20) for each semantic rule, and select the best one. Note that, if all of the keywords in a keyword set are noun, each keyword will generate a semantic rule $\{keyword, null, null\}$.

Eventually, CHAMP generates 320 semantic rules for the 26 undesired behaviors in total (the list of rules can be found in [26]), in which 136 semantic rules are for English comments and 184 semantic rules are for Chinese comments. Note that there are no comments related to the behavior of "modify browser setttings" and thus we use the description in the related policies to extract semantic rules (4 rules in total). The major differences between Chinese comment rules and English comment rules are *synonyms*. Synonyms in Chinese are more frequently used than in English, leading to more rules for some undesired behaviors. For example, two keywords "uninstall" and "remove" of the semantic rules for behavior "fail to uninstall" are generated in English comments, while CHAMP has extracted 5 synonyms of these two keywords in Chinese comments. Table IV shows representative semantic rules for 3 undesired behaviors in English comments (the complete set of rules can be found at Github [26]). As our semantic rules are trained to detect similar sentences that describe the behaviors in the policies, thus the detected sentences are all high quality, which will be evaluated in § V.

D. Semantic Rule Checking

Based on these semantic rules, CHAMP classifies each comment into a type of *UBComments* or others. Given a comment, CHAMP first removes the stopwords and performs word segmentation [52] to extract words from the comment. CHAMP then applies the semantic rules one by one to determine whether the comment matches any rules. It searches the extracted words to see whether the keywords appear in the extracted words and checks the order and distance of successful matching keywords to determine whether they meet the constraints of the semantic rules. As shown in Fig. 1, the motivating app violates two behaviors, i.e., "ads in notification bar" and "virus". Based on the rules defined in Table IV, CHAMP determines that the first comment "too many ads, ..., the notification bar is full of ads" matches 2 semantic rules of the undesired behavior "notification bar", since the comment has the keywords of "notification" , "ads" and "full". Similarly, the other comment contains the keyword of "virus" and thus matches the undesired behavior "virus".

IV. STUDY DESIGN

A. Research Questions

We seek to answer the following research questions (RQs):

RQ1 **How effective is CHAMP in detecting *UBComments*?** As we aim to apply CHAMP to extract *UBComments* in the wild, It is necessary to first evaluate the effectiveness of CHAMP on extracting undesired behaviors using a benchmark dataset.

RQ2 **What kinds of undesired behaviors can be perceived by users?** It is important to explore to what extent we can infer undesired behaviors from user comments, and which behaviors can be perceived by users.

RQ3 **How well do the policies in each app market capture the undesired behaviors reflected by user comments?** As each app market has its own policies, we want to know whether they are effective in flagging undesired behaviors during the app vetting process. App markets with weak app vetting processes are more likely to be exploited.

B. Dataset

1) Collecting App Candidates: To answer the RQs, we first need to harvest a comprehensive dataset that covers as many undesired behaviors as possible. We take advantage of existing efforts, and use a large-scale Android app repository [1]. This repository contains over 6.2 million app items collected from Google Play and 17 third-party app markets. The dataset also provides the detection result of VirusTotal [25], a malware analysis service that aggregates over 60 anti-virus (AV) engines. To better understand the distribution of *UBComments* across apps with different maliciousness levels, we classified our app candidates into 3 categories: malware, grayware and benign apps. As previous studies [53] suggested that some AV engines may not always report reliable results, we regard the apps labeled by over half of the AV engines (>30) as malware, which is supposed to be a reliable threshold by previous work [53]. We consider apps flagged by no AV engines as benign apps, and the other apps as grayware. This roughly classification of malware and grayware might not be accurate enough, but this is not the focus of this paper. As the number of reported engines can be used as an indicator of the maliciousness of the apps, we only want to study the diversity across apps with

TABLE V
OVERVIEW OF OUR COMMENT DATASET.

Market	Malware		Grayware		Benign Apps	
	# apps	# comments	# apps	# comments	# apps	# comments
360 Market [28]	625	33,432	399	205,383	457	161,286
Huawei [29]	144	11,193	388	212,452	296	84,221
Lenovo [30]	184	4,545	252	34,897	225	23,976
Meizu [31]	232	6,766	256	181,212	201	139,662
Oppo [32]	134	16,765	163	503,574	94	76,295
Vivo [33]	196	18,894	266	211,453	295	85,996
Xiaomi [34]	297	32,343	111	177,852	64	60,571
Tencent Myapp [35]	1117	69,044	477	250,649	481	131,949
Google Play [27]	NA	NA	253	183,256	556	311,795
Total	2,027	192,982	1,416	1,960,728	1,713	1,075,751

different levels of maliciousness. We randomly selected 10,000 target app candidates (8,400 Chinese apps and 1,600 Google Play apps) from the dataset of Wang et al. [1], including 4,000 malware, 3,000 grayware and 3,000 benign apps. Note that the 1,600 Google Play apps include 1,000 benign apps and 600 grayware, as all the malware samples were removed by Google Play and we cannot get their comments (NA in Table V).

2) Harvesting the User Comments: All the app markets we studied only provide a limited number of user comments. For example, Google Play review collection service [54] only allows reviews of last week to be crawled for each app. Instead, we built the comment dataset using two alternative approaches. For the 8,400 apps we selected from the Chinese markets, we resort to a third-party app monitoring platform named Kuchuan[55], which has maintained the app metadata including comments from all the Chinese markets we studied. For the 1,600 apps from Google Play, we developed an automated tool to continuously fetch the user comments *everyday* within the span of 3 months. Table V shows the distribution of collected comments. In total, we have collected over 3.2 million user comments from 5,156 apps[1], including 192,982 comments from 2,027 malware, 1,960,728 comments from 1,416 (including 1,163 Chinese apps and 253 Google Play apps) grayware and 1,075,751 comments from 1,713 (including 1,157 Chinese apps and 556 Google Play apps) benign apps. This dataset will be used in the large-scale measurement study (see § VI).

V. EVALUATION OF CHAMP

A. Benchmark Datasets

We curated two benchmark datasets (English and Chinese) to evaluate CHAMP. We first select the apps which are confirmed to have undesired behaviors in the training dataset (see § III-B). For each app, we exclude the comments already used in training dataset. At last, two authors of this paper manually inspected and labelled these comments. Within our affordable efforts, we aim to collect and label 50 comments for each undesired behavior, except for some behaviors with few related apps. Figure 3 shows the distribution of our benchmark (901 Chinese comments and 618 English comments). Note that we cannot find comments for the behavior "browser setting alteration".

[1]Note that, for the selected 10K app candidates, over 4,000 of them have no user comments or very few user comments, which were discard by us.

B. RQ1: Effectiveness of CHAMP

1) Overall Results: Table VI shows the evaluation results. ***It shows that* CHAMP *is very effective in identifying UBComments.*** The average precision and recall are 95% and 93% for the Chinese benchmark, and 97% and 98% for the English benchmark. In particular, CHAMP achieves 90+% of precision and recall for 20 out of 26 types of *UBComments*.

2) False Positives/Negatives: We further manually analyze the mis-classified comments and obtain two observations. First, *the false negatives are colloquial expressions instead of phrases.* For example, the comment "A window of card application pops up continuously" is describing the behavior "ad disruption". But the key phrase "ad" is not in it. Moreover, if we add a new semantic rule with the phrases "window" or "pop up", it may lead to other false positives. Second, *the false positives are generated owing to our insufficiently conservative rules*. For example, the comment "The app is completely useless, btw I thought that this built-in app can not be uninstalled, but it succeeded." is irrelevant to undesired behaviors. However, it is classified to the behavior "fail to uninstall" since it has the phrases "can not" and "uninstall". Analogously, if we upgrade our rules to be more conservative, it may lead to more false negatives. These are the inherent limitations of rule-based matching methods. We will further discuss it in § VII.

3) Comparison with Text Similarity Approach: We compare CHAMP with the text similarity approach, which classifies a comment to a type of undesired behavior based on text similarity between the comment and the classified comments in the training dataset (see § III-B). We regard the behavior with the highest similarity score as the classification result.

As shown in Table VI, CHAMP achieves significantly better results than the text similarity approach. The average precision and recall achieved by the text similarity approach are 85% and 81% (v.s. 95% and 93% achieved by CHAMP) for the Chinese comment dataset, and 77% and 85% (v.s. 97% and 98% achieved by CHAMP) for the English comment dataset, respectively. In particular, CHAMP outperforms the text similarity approach on all behaviors. Such results indicate that the order and distance constraints adopted by our semantic rules can greatly reduce the false positives/negatives. For example, the comment "I can not install the app" is similar to "I installed but it can not help me back up files" considering their text similarity, but they are describing different types of undesired behaviors. CHAMP correctly distinguishes these two comments while the text similarity approach classifies both of them to the same type of undesired behavior.

VI. LARGE-SCALE MEASUREMENT STUDY

A. RQ2: UBComments in the Wild

1) Overall Results: From the dataset we harvested (see § IV), **CHAMP identifies 94,028 *UBComments*, belonging to 2,440 apps (47%)**. Each app has received 39 *UBComments* from multiple users on average. This indicates that *UBComments* are prevalent in the mobile app ecosystem, and the users who are sensitive to those policy violations are

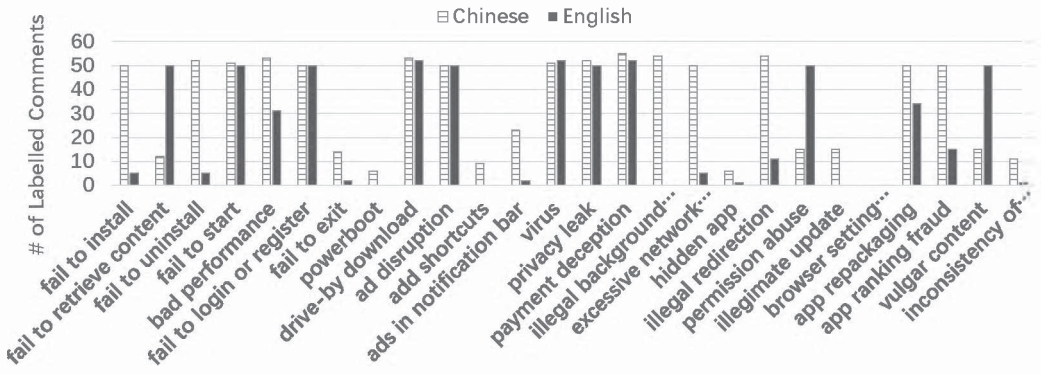

Fig. 3. Distribution of labelled benchmarks.

TABLE VI
EVALUATION RESULTS ON THE BENCHMARK DATASETS (BEST RESULTS ARE SHOWN IN BOLD).

Category	Behavior	Benchmark (Chinese)						Benchmark (English)					
		CHAMP			Similarity-Based Tool			CHAMP			Similarity-Based Tool		
		precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
Functionality and Performance	fail to install	**96%**	**94%**	**95%**	87%	90%	88%	**83%**	**100%**	**91%**	71%	100%	83%
	fail to retrieve content	**100%**	**100%**	**100%**	89%	67%	76%	**96%**	**98%**	**97%**	85%	80%	82%
	fail to uninstall	**100%**	**96%**	**98%**	91%	92%	91%	**100%**	**100%**	**100%**	63%	100%	77%
	fail to start (e.g., crash)	**98%**	**96%**	**97%**	88%	88%	88%	**96%**	**96%**	**96%**	85%	82%	84%
	bad performance (e.g., no responding)	**88%**	**94%**	**91%**	86%	91%	88%	**91%**	**97%**	**94%**	80%	77%	79%
	fail to login or register	**98%**	**98%**	**98%**	87%	90%	88%	**96%**	**100%**	**98%**	87%	90%	88%
	fail to exit	**93%**	**93%**	**93%**	81%	93%	87%	**100%**	**100%**	**100%**	100%	100%	100%
	powerboot	**83%**	**83%**	**83%**	83%	83%	83%	NA	NA	NA	NA	NA	NA
Advertisement	drive-by download	**100%**	**94%**	**97%**	75%	85%	80%	**100%**	**98%**	**99%**	73%	73%	73%
	ad disruption	**100%**	**100%**	**100%**	73%	64%	68%	**100%**	**100%**	**100%**	69%	70%	69%
	add shortcuts in launching menu	**100%**	**100%**	**100%**	100%	78%	88%	NA	NA	NA	NA	NA	NA
	ads in notification bar	**96%**	**96%**	**96%**	55%	96%	70%	**100%**	**100%**	**100%**	50%	100%	67%
Security	virus	**100%**	**98%**	**99%**	100%	86%	93%	**100%**	**100%**	**100%**	100%	88%	94%
	privacy leak	**98%**	**94%**	**96%**	88%	85%	86%	**96%**	**96%**	**96%**	85%	82%	84%
	payment deception	**100%**	**91%**	**95%**	92%	87%	90%	**98%**	**96%**	**97%**	91%	79%	85%
	illegal background behavior (e.g., sms)	**91%**	**91%**	**91%**	73%	76%	75%	NA	NA	NA	NA	NA	NA
	excessive network traffic	**98%**	**98%**	**98%**	90%	90%	90%	**100%**	**100%**	**100%**	80%	80%	80%
	hidden app	**100%**	**100%**	**100%**	100%	67%	80%	**100%**	**100%**	**100%**	100%	100%	100%
	illegal redirection	**88%**	**85%**	**87%**	88%	78%	82%	**92%**	**100%**	**96%**	75%	82%	78%
	permission abuse	**92%**	**80%**	**86%**	86%	80%	83%	**100%**	**96%**	**98%**	89%	84%	87%
	illegitimate update (e.g., update to other app)	**87%**	**87%**	**87%**	86%	80%	83%	NA	NA	NA	NA	NA	NA
	browser setting alteration	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Illegitimate Behavior of Developers	app repackaging	**85%**	**92%**	**88%**	78%	86%	82%	**97%**	**100%**	**99%**	65%	65%	65%
	app ranking fraud	**96%**	**98%**	**97%**	83%	80%	82%	**83%**	**100%**	**91%**	69%	73%	71%
Content	vulgar content (e.g., pornography, anti-society)	**100%**	**87%**	**93%**	85%	73%	79%	**100%**	**92%**	**96%**	95%	84%	89%
	inconsistency between functionality and description	**100%**	**91%**	**95%**	83%	45%	59%	**100%**	**100%**	**100%**	33%	50%	40%

TABLE VII
DISTRIBUTION OF *UBComments* BY CATEGORIES.

Category	#Comment (%)	#App (%)
Functionality/Performance	57,541 (61%)	1701 (70%)
Advertisement	22,885 (24%)	1023 (42%)
Security	13,765 (15%)	1098 (45%)
Illegitimate Behavior	1,129 (1%)	173 (7%)
Content	536 (0.5%)	72 (3%)
Total	94,028	2,440

willing to report them in the comments. Table VII shows the distribution of *UBComments* and apps across different categories of behaviors. Over 61% of the *UBComments* and over 70% of the corresponding apps were complained to have "functionality and performance" issues. This shows that users are most sensitive to the issues that directly affect their uses of the apps. **For the 26 behaviors we summarized, 25 of them could be perceived by users.** The most popular behaviors of *UBComment* are "fail to start", "ad disruption", and "payment deception", accounting for 79.4% of the *UBComments*. Both "fail to start" and "ad disruption" are related to user experiences, while "payment deception" shows users' security concerns.

Manual Verification of Undesired Behaviors. To analyze whether the undesired behaviors described in user comments reflect the real behaviors of mobile apps, we make effort to perform a manual verification here. For each of the 25 identified perceived behaviors, we randomly select three apps (75 apps in total) and manually verify if indeed the apps violated the policies as described. Our manual verification follows a series of steps. We first install them on smartphones to see whether they have shown undesired behaviors as user complained (e.g., ad disruption and malicious behaviors, etc.). Then we rely on Testin [56], a service that provides app testing on thousands of real-world smartphones, to check the functionality and performance issues (e.g., fail to start and fail to install). Furthermore, we leverage static analysis

TABLE VIII
DISTRIBUTION OF *UBComments* BY RATING STARS.

Dataset	1-star	2-star	3-star	4-star	5-star
Malware	28.14%	17.70%	9.09%	3.85%	2.16%
Chinese Grayware	23.36%	16.22%	9.23%	4.37%	0.64%
Chinese Benign Apps	25.38%	17.58%	10.07%	5.95%	0.96%
GPlay Grayware	6.50% (14.28%)	0.05% (0.07%)	0.03% (0.04%)	0.02% (0.02%)	0% (0%)
GPlay Benign Apps	1.96% (8.64%)	0.33% (0.92%)	0.15% (0.25%)	0.07% (0.10%)	0.01% (0.02%)
Total	19.45%	12.15%	6.69%	2.78%	0.71%

tools (e.g., LibRadar [57] and FlowDroid [58]) to extract and inspect behavior-related app information (e.g., sensitive code, permissions and libraries). At last, for the apps in behavior "app ranking fraud", we compare their comments based on existing approaches proposed in [59], [60] to find fake comments. Overall, 72 apps (96%) have been confirmed with the undesired behaviors as user commented. For the 3 unconfirmed cases (one in the vulgar content category, and two in the payment deception category), our dynamic analysis found that their services have stopped and our static analysis failed due to they have adopted heavy obfuscation and code protection using packing services. Nevertheless, we show that *most of the undesired behaviors can be confirmed.*

2) Low-rating Comments vs. High-rating Comments (RQ2.1): We study the distribution of *UBComments* across comments with different ratings (from 1-star to 5-star).

Quantitative Analysis. As shown in Table VIII, it is apparent that **low-rating comments (*i.e.,* 1-star and 2-star) are more likely to describe undesired behaviors**. *UBComments* account for roughly 20% and 12% for the 1-star comments and the 2-star comments, and 2.78% and 0.71% for the 4-star and the 5-star comments, respectively. Note that the proportion of *UBComments* in Google Play is much lower than that of Chinese markets. The major reason is that the crawled comments from Google Play contain a large amount of blank comments, *i.e.,* the comments with only a rating but no descriptions. We further eliminate such comments and report the result (see the percentage in brackets in Table VIII).

Qualitative Analysis. As shown in Figure 4, the distributions of *UBComments* in Chinese markets and Google Play show great diversity, and thus we discuss them separately.

For app comments in Chinese markets, the distribution of undesired behaviors does not show much diversity across *UBComments* with different ratings. Behaviors of the "Functionality and Performance" and "Advertisement" types are most prevalent across all the ratings, with the "Fail to start" and "Ad disruption" types are quite noticeable. Moreover, we find that security related behaviors are prevalent in both low-rating and high-rating comments of malware, but only prevalent in low-rating comments of grayware and benign apps. It is quite surprising that users complain about the security issues (*e.g.,* payment deception) but give the app (malware) a high rating. Thus, we make efforts to manually examine all such "contradictory" comments (21,859 in total), and identify two major reasons. First, the default comment rating of most Chinese app markets is 5-star, thus a number of users may only complain the app in the comments but forget to assign a rating. Second, it is quite possible that some users misunderstand the meanings of 1-star and 5-star. For example, we find that several users assign totally opposite ratings in all their comments, *i.e.,*, 1-star with really good comments, but 5-star with negative comments, including the *UBComments*. It suggests the poor knowledge of the rating system for market users, and the new challenges in analyzing the comments of third-party app markets. Nevertheless, CHAMP can reveal how the users feel about their experiences, and even could improve the techniques of app risk assessments based on user comments [61], [20].

In Google Play, the distribution of *UBComments* in low-rating comments and high-rating comments are quite different. Users generally give 1-star in their comments when they find undesired behaviors in the app, even if the behaviors do not belong to the "security" category. We only find a few comments that are related to the "vulgar content" type in other comments. This might be due to the high-quality market which pays more attention to policy regulations, and this more mature and regulated ecosystem enables users to better comprehend the ratings when providing comments.

3) Malware vs. Grayware vs. Benign Apps (RQ2.2): For Chinese markets, over 42% of malware samples have *UBComments*, and they have occupied 7% of the comments. As a contrast, over 57% of benign app samples and 57% of grayware samples have *UBComments*, and the percentages of these comments are 4% and 3%, respectively. For Google Play, over 32% of benign apps and 38% of grayware apps have *UBComments*, and they account for 0.3% and 1% of the overall comments (0.6% and 1.5% after removing the empty ones from the overall comments), respectively. In general, one would think that malicious apps have more UBComments than gray and benign apps, as their behaviors are more likely to inconsistent with users' expectation. However, the results are different for what we expected, i.e., the percentage of *UBComments* does not show much difference across malware, grayware and benign apps. There are mainly two reasons. First, the policy-violation behaviors of two major types, "Functionality and Performance" and "Advertisement", are prevalent in both malicious and benign apps, *e.g.,* over 74% of the UBComments in third-party benign apps are related to "Functionality and Performance". Second, some malware samples were removed in time by markets, and thus malicious apps have not received much complaints than expected. Note that the security-related undesired behaviors show different distributions across malicious, gray, and benign apps (see Figure 4). As to Chinese markets, over 27% of the *UBComments* belong to the security category for malware, while the percentages for grayware and benign apps are 14% and 9%. As to Google Play, over 31% of the *UBComments* in grayware are security related (V.S. 16% in benign apps). Furthermore, we observe that **many user-perceived undesired behaviors (including security-related ones) were found in both malware and "benign apps". It suggests that some**

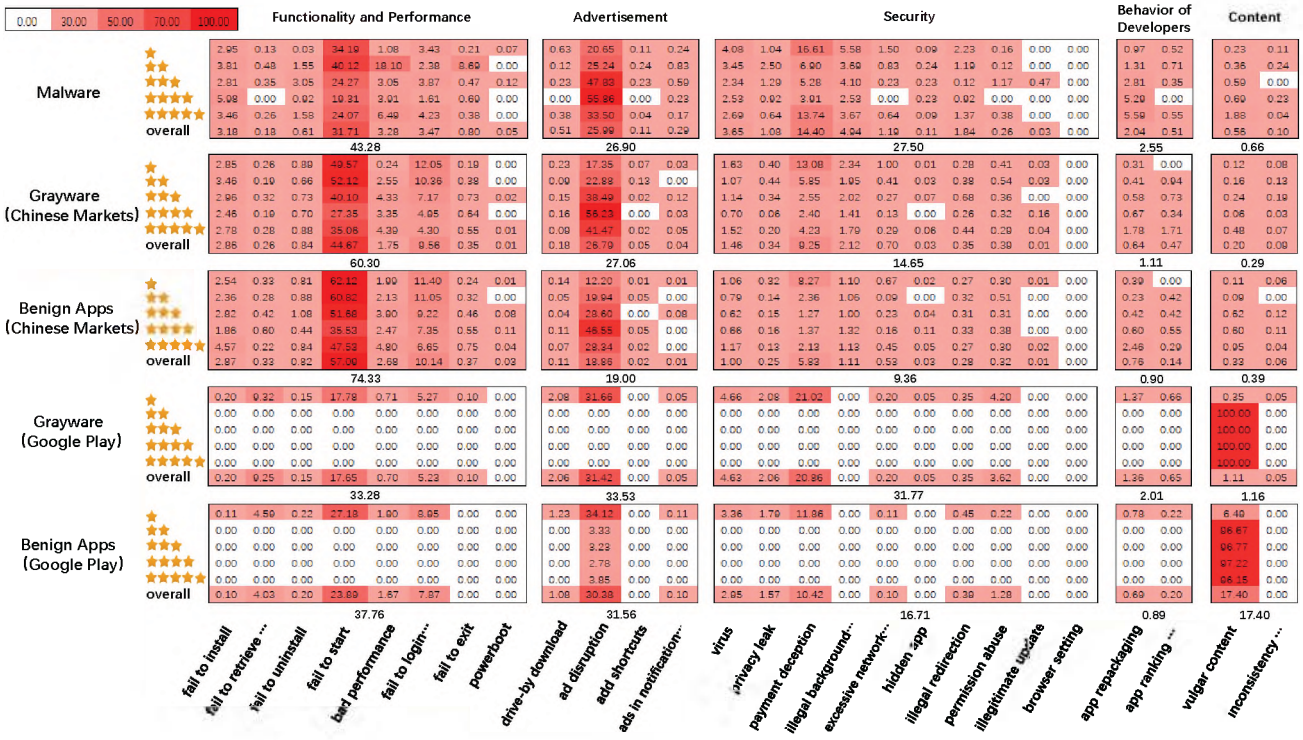

Fig. 4. Distribution of *UBComments* across different ratings (y-axis) and undesired behaviors (x-axis) in different app categories. Each row adds up to 100%, with each cell representing the percentage of a specific undesired behavior in the *UBComments* for a specific rating (*e.g.,* 5 stars) in an app category (*e.g.,* Malware). The depth of the color is also used to indicate the percentage in the cell, *i.e.,*, a deep color indicating a large percentage. For each app category, a behavior category box represents the *UBComments* of a specific behavior category (*e.g.,* Security), and the number below the box shows the percentage of the *UBComments* in that behavior category.

malicious behaviors are hard to detect by AV engines but user comments could provide insights for capturing them.

B. RQ3: Undesired Behaviors Across Markets

We perform market-level analysis to investigate the differences across markets. On one hand, for the undesired behaviors declared in the policies of each market, we seek to measure how many such behaviors have been identified in our dataset. This result could be used to measure the effectiveness of market regulation, *i.e.,* how many of these undesired apps have bypassed the corresponding auditing process. On the other hand, for other undesired behaviors that were not declared in the policies of a market, we seek to explore whether we could find such behavior related comments in the corresponding markets.

Table II shows the results. Roughly 34% to 65% of the apps (the numbers in bold) from each market have found comments for undesired behaviors described in each of their market policies. Over 65% of the apps in Huawei Market have violated its market policies, while the percentage of such apps in Google Play is 34%. From another point of view, roughly 5% to 60% of the apps (besides Google Play, as it covers all the behaviors we summarized in this paper) have been complained of having undesired behaviors that are not captured by the markets' policies. For example, over 60% of the apps in the 360 Market have undesired behaviors that are not listed in its market policies. **This may open doors for malicious developers to exploit the insufficient vetting process.**

VII. Discussions

A. Relation with Program Analysis

A large number of papers were focused on using program analysis to detect the security [62], [63], [64], privacy [65], [66], [38], [67], [68], ads/third-party library [57], [9], [69], [11], and functionality issues [70], [71], [72], [73], [74] of mobile apps. In contrast, this paper focuses on a different perspective, i.e., *how the users feel about their experiences*. Users' expectations play a big role on how much the users can tolerate the apps' behaviors.

First, although program analysis could be adopted to identify whether some sensitive behaviors exist in mobile apps, **it is non-trivial to verify whether the behaviors violate the policy. The borderline between policy-violation and tolerable misbehaviors is fuzzy and highly dependent on users' subjective expectations**. For example, program analysis can easily identify ad libraries used in apps. However, aggressive mobile ads cannot be simply conflated with the detection of ad libraries. The detection of ad libraries, enabled by program analysis techniques, cannot take what users really feel about the ads into consideration. Second, **a number of the policy-violation behaviors, *e.g.,* payment deception and**

vulgar content, are difficult to be triggered and detected by program analysis techniques. However, they are indeed much easier revealed by user comments.

Thus, CHAMP is complementary to program analysis, which can provide insight to identify the boundary between policy-violation behaviors and tolerable misbehaviors. Instead of identifying the policy-violation behaviors directly, CHAMP can serve as a *whistle blower* that assigns policy-violation scores and identifies most informative comments for apps (*e.g.,* putting security related comments at top). Note that, not all the apps with UBComments should be removed by the app market. App vetting is aimed at promoting the overall quality of apps in the market. Thus, app markets would generally give developers warnings and buffer time to fix undesired behaviors in their apps (rather than removing them directly). With the help of CHAMP, it will be possible to pinpoint more urgent violations accurately, such as security-related ones, so that the markets could choose their reaction accordingly.

B. Threats to Validity

First, the taxonomy we summarized may be incomplete. Although we have manually summarized 26 undesired behaviors, our taxonomy may still be incomplete since it was built based on current policies. However, our approach is generic and can be reused to support the detection of new types of undesired behaviors. *Second, our approach inherits the drawbacks of rule-based approaches.* Though our approach was proven to be quite effective during our evaluation, the semantic rules we summarized may not be complete and could introduce false positives/negatives as mentioned in Section V-A. Nevertheless, market policies are rarely updated. Furthermore, our approach has strong expansibility of extracting new semantic rules for emerging app store policies. When policies evolve, new training can be performed to obtain new rules. Note that only the training process is semi-automated, as we need to manually label the classified comments. Our rules are extracted automatically from the labelled comments, which can be applied to identify UBComments automatically. Third, *we are not able to verify all the undesired behaviors for all the apps we identified*. We only sample 75 apps for manual verification, and found 96% of them can be confirmed. We found *most of the behaviors cannot be easily identified using automated tools, that is the reason why UBComments are prevalent even though these apps have already passed the market vetting process*. This motivates the research community to develop better tools for identifying such behaviors. Nevertheless, as aforementioned (see Section VII-A), instead of identifying the policy-violation behaviors directly, CHAMP could raise alarm based on the number of undesired comments and reported users.

VIII. Related Work

To the best of our knowledge, our paper is the first one that identifies undesired behaviors from user comments. Nevertheless, there are a number of studies focusing on app comments from different perspectives. We present and discuss briefly related works on (1) general app comment analysis, and (2) using NLP techniques in mobile app analysis.

App Comment Analysis. Mobile app comments have been extensively studied from other perspectives, including mining user opinions [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], app comment filtering [88], [79], [89], and exploring other concerns [90], [91], [92], [93], [94]. For example, Chen *et al.* [88] pioneered the prioritization of user comments with AR-Miner. Chen *et al.* [91] conducted a study on the unreliable maturity content ratings of mobile apps, which will result in inappropriate risk exposure for the children and adolescents. Nguyen *et al.* [90] proposed to analyze the relationship between user comments and security-related changes in Android apps. Kong *et al.* [20] presented a machine-learning technique to identify 4 pre-defined types of security-related comments. Although app comments have been extensively studied from other perspectives, none of the above work correlates user comments to the undesired behaviors described in market policies and none of them can be easily adopted/extended to study this issue.

NLP in Mobile App Analysis. Besides user comments, NLP techniques have been widely adopted to study app descriptions, privacy policies, and other meta text information related to mobile apps. Whyper [38] and Autocog [95] adapt NLP techniques to characterize the inconsistencies between app descriptions and declared permissions. PPChecker [96] is a system for identifying the inconsistencies between privacy policy and the sensitive behaviors of apps. CHABADA [97] adapts NLP techniques to cluster apps using description topics, and then identifies the outliers of API usage within each cluster. Our work is the first to investigate the correlation between user comments and market policies.

IX. Conclusion

We present the first large-scale study to investigate the correlation between user comments and market policies. In particular, we propose CHAMP, a semantic-rule based approach that effectively identifies *UBComments*. We apply CHAMP to a large scale user comment dataset and observe that *UBComments* are prevalent in the ecosystem, even though app markets explicitly declared their policies and applied extensive vetting. CHAMP offers a promising approach to detect policy violations, so as to help market maintainers identify these violations timely and further improve the app vetting process.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (grant numbers 62072046, 61702045 and 61772042), NSF (CNS-1755772), and Hong Kong RGC Projects (No. 152223/17E, 152239/18E, CityU C1008-16G).

References

[1] H. Wang, Z. Liu, J. Liang, N. Vallina-Rodriguez, Y. Guo, L. Li, J. Tapiador, J. Cao, and G. Xu, "Beyond google play: A large-scale comparative study of chinese android app markets," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 293–307.

[2] H. Wang, H. Li, and Y. Guo, "Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play," in *The World Wide Web Conference*, 2019, pp. 1988–1999.

[3] "2018 Malware Forecast: the onward march of Android malware," 2018, https://nakedsecurity.sophos.com/2017/11/07/2018-malware-forecast-the-onward-march-of-android-malware/.

[4] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 76, 2017.

[5] Y. Tang, Y. Sui, H. Wang, X. Luo, H. Zhou, and Z. Xu, "All your app links are belong to us: understanding the threats of instant apps based attacks," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 914–926.

[6] Y. Hu, H. Wang, Y. Zhou, Y. Guo, L. Li, B. Luo, and F. Xu, "Dating with scambots: Understanding the ecosystem of fraudulent dating applications," *arXiv preprint arXiv:1807.04901*, 2018.

[7] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu *et al.*, "Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2421–2436.

[8] Y. Hu, H. Wang, R. He, L. Li, G. Tyson, I. Castro, Y. Guo, L. Wu, and G. Xu, "Mobile app squatting," in *Proceedings of The Web Conference 2020*, 2020, pp. 1727–1738.

[9] F. Dong, H. Wang, L. Li, Y. Guo, T. F. Bissyandé, T. Liu, G. Xu, and J. Klein, "Frauddroid: Automated ad fraud detection for android apps," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 257–268.

[10] B. Andow, A. Nadkarni, B. Bassett, W. Enck, and T. Xie, "A study of grayware on google play," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 224–233.

[11] T. Liu, H. Wang, L. Li, X. Luo, F. Dong, Y. Guo, L. Wang, T. Bissyandé, and J. Klein, "Maddroid: Characterizing and detecting devious ad contents for android apps," in *Proceedings of The Web Conference 2020*, 2020, pp. 1715–1726.

[12] T. Liu, H. Wang, L. Li, G. Bai, Y. Guo, and G. Xu, "Dapanda: Detecting aggressive push notifications in android apps," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 66–78.

[13] H. Zhou, H. Wang, Y. Zhou, X. Luo, Y. Tang, L. Xue, and T. Wang, "Demystifying diehard android apps," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 187–198.

[14] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, "Why are android apps removed from google play?: a large-scale empirical study," in *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 2018, pp. 231–242.

[15] Google, "Google play developer policy," https://play.google.com/about/developer-content-policy.

[16] "Google Bouncer," 2018, https://krebsonsecurity.com/tag/google-bouncer/.

[17] "Combating Potentially Harmful Applications with Machine Learning at Google: Datasets and Models," 2018, https://android-developers.googleblog.com/2018/11/combating-potentially-harmful.html.

[18] H. Wang, J. Si, H. Li, and Y. Guo, "Rmvdroid: towards a reliable android malware dataset with app metadata," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 404–408.

[19] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in *Proceedings of the Seventh European Workshop on System Security*. ACM, 2014, p. 5.

[20] D. Kong, L. Cen, and H. Jin, "Autoreb: Automatically understanding the review-to-behavior fidelity in android applications," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, 2015, pp. 530–541.

[21] G. Xiaodong and K. Sunghun, ""what parts of your apps are loved by users?" (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 760–770.

[22] Google, 2019, https://developer.android.com/distribute/best-practices/launch/launch-checklist.

[23] G. Cuiyun, Z. Jichuan, L. Michael, R, and K. Irwin, "Online app review analysis for identifying emerging issues," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 48–58.

[24] Y. Li, B. Jia, Y. Guo, and X. Chen, "Mining user reviews for mobile app comparisons," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 75:1–75:15, Sep. 2017.

[25] "VirusTotal," 2018, https://www.virustotal.com.

[26] "UBCFinder," 2020, https://github.com/UBCFinder/UBCFinder.

[27] "Google Play," 2019, https://play.google.com/intl/zh-CN/about/developer-content-policy/#!?modal_active=none.

[28] "360 Market," 2018, http://zhushou.360.cn/.

[29] "Huawei," 2018, http://app.hicloud.com/.

[30] "Lenovo," 2018, https://www.lenovomm.com/apps/1038/0?type=1.

[31] "Meizu," 2018, http://app.meizu.com/.

[32] "Oppo," 2018, https://store.oppomobile.com/.

[33] "Vivo," 2018, http://zs.vivo.com.cn/.

[34] "Xiaomi," 2018, http://app.mi.com/.

[35] "Tencent Myapp," 2018, https://sj.qq.com/myapp/.

[36] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie, "Automated extraction of security policies from natural-language software documents," in *International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 12:1–12:11.

[37] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in *International Conference on Software Engineering (ICSE)*, 2012, pp. 815–825.

[38] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "{WHYPER}: Towards automating risk assessment of mobile applications," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 527–542.

[39] C. Aggarwal and C. Zhai, *Mining text data*, ser. A survey of text clustering algorithms. Boston, MA: Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-1-4614-3223-4_4

[40] Y. Li, D. McLean, A. Bandar, Zuhair, K. Crockett, and et al, "Sentence similarity based on semantic nets and corpus statistics," *IEEE Transactions on Knowledge & Data Engineering*, no. 8, pp. 1138–1150, 2006.

[41] A. Islam and D. Inkpen, "Semantic text similarity using corpus-based word similarity and string similarity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 2, p. 10, 2008.

[42] X. Yan, J. Guo, Y. Lan, and et al, "A biterm topic model for short texts," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 1445–1456.

[43] C. Weizheng, W. Jinpeng, Z. Yan, Y. Hongfei, and X. L, "User based aggregation for biterm topic model," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, 2015, pp. 489–494.

[44] W. Jian, G. Panpan, M. Yutao, H. Keqing, and et al, "A web service discovery approach based on common topic groups extraction," *IEEE Access*, no. 5, pp. 10 193–10 208, 2017.

[45] L. Xiangsheng, R. Yanghui, X. Haoran, and et al, "Bootstrapping social emotion classification with semantically rich hybrid neural networks," *IEEE Transactions on Affective Computing*, no. 8, pp. 428–428, 2017.

[46] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '99. New York, NY, USA: ACM, 1999, pp. 50–57. [Online]. Available: http://doi.acm.org/10.1145/312624.312649

[47] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944937

[48] Wikipedia, "Tf-idf," http://en.wikipedia.org/wiki/Tf-idf.

[49] "Chinese Stopwords List," 2018, https://github.com/goto456/stopwords.

[50] "English Stopwords List," 2018, https://github.com/stopwords-iso/stopwords-en.

[51] "Part of speech," 2020, https://en.wikipedia.org/wiki/Part_of_speech.

[52] "Word segmentation Library," 2018, https://pypi.org/project/jieba/.

[53] F. Wei, Y. Li, S. Roy, and et al, "Deep ground truth analysis of current android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 252–276.

[54] "Google play reviews collection service," 2018, https://developers.google.com/android-publisher/api-ref/reviews.

[55] "Kuchuan," 2018, http://www.kuchuan.com/.

[56] Testin, "Testin service," https://www.testin.cn/.

[57] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: fast and accurate detection of third-party libraries in android apps," in *Proceedings of the 38th international conference on software engineering companion*, 2016, pp. 653–656.

[58] A. Steven, R. Siegfried, F. Christian, B. Eric, B. Alexandre, K. Jacques, L. T. Yves, O. Damien, and M. Patrick, "Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *ACM Sigplan Notices*, no. 6, pp. 259–269, 2014.

[59] Y. Hu, H. Wang, L. Li, Y. Guo, G. Xu, and R. He, "Want to earn a few extra bucks? a first look at money-making apps," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 332–343.

[60] M. Daniel and M. Walid, "Towards understanding and detecting fake reviews in app stores," *Empirical Software Engineering*, pp. 1–40, 2019.

[61] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Mobile app recommendations with security and privacy awareness," in *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, 2014, pp. 951–960.

[62] W. Pengcheng, S. Jeffrey, W. Yanzhao, and et al, "Ccaligner: a token based large-gap clone detector," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 1066–1077.

[63] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1025–1035.

[64] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, "Graph embedding based familial analysis of android malware using unsupervised learning," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 771–782.

[65] H. Wang, J. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 1107–1118.

[66] M. Liu, H. Wang, Y. Guo, and J. Hong, "Identifying and analyzing the privacy of apps for kids," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, 2016, pp. 105–110.

[67] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 426–436.

[68] H. Wang, Y. Li, Y. Guo, Y. Agarwal, and J. I. Hong, "Understanding the purpose of permission use in mobile apps," *ACM Transactions on Information Systems (TOIS)*, vol. 35, no. 4, pp. 1–40, 2017.

[69] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 71–82.

[70] L. Li, T. F. Bissyandé, H. Wang, and J. Klein, "Cid: Automating the detection of api-related compatibility issues in android apps," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 153–163.

[71] L. Wei, Y. Liu, S.-C. Cheung, H. Huang, X. Lu, and X. Liu, "Understanding and detecting fragmentation-induced compatibility issues for android apps," *IEEE Transactions on Software Engineering*, 2018.

[72] T. Shin Hwei, D. Zhen, G. Xiang, and et al, "Repairing crashes in android apps." IEEE, 2018, pp. 187–198.

[73] B. Pan, L. Bin, S. Wenchang, and et al, "Nar-miner: Discovering negative association rules from code for bug detection," in *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 411–422.

[74] H. Wang, H. Liu, X. Xiao, G. Meng, and Y. Guo, "Characterizing android app signing issues," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 280–292.

[75] K. H, "On identifying user complaints of ios apps," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1474–1476.

[76] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about? a study on free ios apps," *IEEE Software*, vol. 99, no. 1, pp. 1–10, 2014.

[77] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *IEEE 22nd international requirements engineering conference (RE)*. IEEE, 2014, pp. 153–162.

[78] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 582–591.

[79] B. Fu, J. Lin, L. Li, and et al, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 1276–1284.

[80] A. Di, Sorbo, S. Panichella, V. Alexandru, C, and et al., "Surf: summarizer of user reviews feedback," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 55–58.

[81] L. Villarroel, G. Bavota, B. Russo, and et al, "Release planning of mobile apps based on user reviews," in *2016 IEEE/ACM 38th International Conference on Software Engineering*. IEEE, 2016, pp. 14–24.

[82] S. Panichella, A. Di, Sorbo, G. E, and et al, "Ardoc: app reviews development oriented classifier," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 1023–1027.

[83] M. Vu, P, V. Pham, H, and T. Nguyen, T, "Phrase-based extraction of user opinions in mobile app reviews," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 726–731.

[84] Y. Li, B. Jia, Y. Guo, and X. Chen, "Mining user reviews for mobile app comparisons," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 75:1–75:15, Sep. 2017.

[85] S. FA, S. Kairit, and P. Dietmar, "Using app reviews for competitive analysis: tool support," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*. ACM, 2019, pp. 40–46.

[86] D. Jacek, L. Emmanuel, P. Anna, and S. Angelo, "Finding and analyzing app reviews related to specific features: A research preview," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2019, pp. 183–189.

[87] E. Noei, F. Zhang, and Y. Zou, "Too many user-reviews, what should app developers look at first?" *IEEE Transactions on Software Engineering*, pp. 1–12, 2019.

[88] N. Chen, J. Lin, S. C. H. Hoi, and et al, "Ar-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 767–778.

[89] L. Washington, V. Felipe, A. Rafael, and et al, "A feature-oriented sentiment rating for mobile app reviews," in *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 2018, pp. 1909–1918.

[90] D. C. Nguyen, E. Derr, M. Backes, and S. Bugiel, "Short text, large effect: Measuring the impact of user reviews on android app security & privacy," 2019.

[91] Y. Chen, H. Xu, Y. Zhou, and et al, "Is this app safe for children?: a comparison study of maturity ratings on android and ios applications," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 201–212.

[92] L. Cen, L. Si, N. Li, and et al, "User comment analysis for android apps and cspi detection with comment expansion," in *Proceeding of the 1 st International Workshop on Privacy-Preserving IR: When Information Retrieval Meets Privacy and Security*. PIR, 2014, pp. 25–30.

[93] F. Palomba, M. Linares-Vasquez, G. Bavota, and et al, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 291–300.

[94] H. Chen, D. He, S. Zhu, and et al, "Toward detecting collusive ranking manipulation attackers in mobile app markets," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 58–70.

[95] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1354–1365.

[96] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 538–549.

[97] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2014.