Mobile App Squatting

Yangyu Hu
Beijing University of Posts and Telecommunications, China

Haoyu Wang*
Beijing University of Posts and Telecommunications, China

Ren He
Beijing University of Posts and Telecommunications, China

Li Li
Faculty of Information Technology, Monash University, Australia

Gareth Tyson
Queen Mary University of London, United Kingdom

Ignacio Castro
Queen Mary University of London, United Kingdom

Yao Guo
MOE Key Lab of HCST, Peking University, China

Lei Wu
Zhejiang University, China

Guoai Xu*
Beijing University of Posts and Telecommunications, China

ABSTRACT

Domain squatting, the adversarial tactic where attackers register domain names that mimic popular ones, has been observed for decades. However, there has been growing anecdotal evidence that this style of attack has spread to other domains. In this paper, we explore the presence of squatting attacks in the mobile app ecosystem. In *"App Squatting"*, attackers release apps with identifiers (e.g., app name or package name) that are confusingly similar to those of popular apps or well-known Internet brands. This paper presents the first in-depth measurement study of app squatting showing its prevalence and implications. We first identify 11 common deformation approaches of app squatters and propose *"AppCrazy"*, a tool for automatically generating variations of app identifiers. We have applied AppCrazy to the top-500 most popular apps in Google Play, generating 224,322 deformation keywords which we then use to test for app squatters on popular markets. Through this, we confirm the scale of the problem, identifying 10,553 squatting apps (an average of over 20 squatting apps for each legitimate one). Our investigation reveals that more than 51% of the squatting apps are malicious, with some being extremely popular (up to 10 million downloads). Meanwhile, we also find that mobile app markets have not been successful in identifying and eliminating squatting apps. Our findings demonstrate the urgency to identify and prevent app squatting abuses. To this end, we have publicly released all the identified squatting apps, as well as our tool AppCrazy.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Human-centered computing → Empirical studies in ubiquitous and mobile computing**.

KEYWORDS

app squatting, typosquatting, fake app, Android, malware

*Haoyu Wang and Guoai Xu are co-corresponding authors.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380243

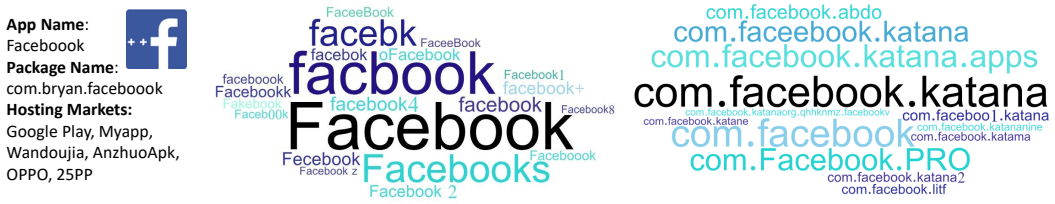
ACM Reference Format:
Yangyu Hu, Haoyu Wang, Ren He, Li Li, Gareth Tyson, Ignacio Castro, Yao Guo, Lei Wu, and Guoai Xu. 2020. Mobile App Squatting. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380243

1 INTRODUCTION

In *domain squatting*, attackers register domain names that resemble legitimate ones to capture traffic intended for the original domain. The most popular domain squatting attack is typosquatting [30], where attackers target mistakes (e.g., common misspellings). These mistakes may lead users to malicious/phishing websites, potentially scooping up millions of unwary visitors. Indeed, a recent report [11] suggested that a simple form of website typosquatting (e.g., changing dot-com to 'dot-cm') attracted roughly 12 million visits in just a quarter of 2018. The effectiveness of domain squatting has led attackers to apply similar attacks to other areas, e.g., to capture emails (due to incorrectly typed recipient addresses) [103], and install malicious code via the famous PyPI (Python Package Index) repository [22].

Following the same postulation, we wonder *if similar kinds of squatting attacks have migrated into the mobile app ecosystem.* Anecdotal evidence suggests that the answer is yes, with a number of popular media articles discussing the threat [14, 15, 35]. For example, attackers managed to distribute their fake "WhatsApp" app to millions of mobile users via the official Google Play Store by simply adding a Unicode-encoded space at the end of WhatsApp's ID [15]. Similarly, a fake app named *Teligram* sneaked into the Google Play pretending to be a new version of the real Telegram app [35]. Naturally, the fake versions often look identical to the legitimate ones, making it extremely difficult for users to distinguish. Although official platforms may provide users with additional warnings, recent studies have suggested that app squatting might be easier to perform on less regulated platforms [72, 77, 84, 85, 93, 106].

To illustrate how app squatting attacks might be effective against a typical user, Figure 1(a) shows an example of a real squatting app which appeared in 6 hosting markets (e.g., Google Play and Myapp). It targets Facebook by sharing a similar app name, package name, and icon. This is not an isolated case. Figure 1(b) and Figure 1(c) present word clouds of a variety of confusingly similar app names and packages that target the Facebook app on Google Play: even



(a) Example of a squatting app **(b)** App name wordcloud **(c)** Package name wordcloud

Figure 1: Examples of squatting apps: (a) A real-world squatting app that targets Facebook; (b) The word cloud of squatting app names of Facebook; (c) The word cloud of squatting package names of Facebook (com.facebook.katana).

for experienced eyes, it can be difficult to pinpoint the differences between the original and the squatting ones.

With this motivation in mind, we present the first detailed study of *App Squatting* in the Android app ecosystem. We start by performing a preliminary study of app squatting on 10 popular apps (**Section 3**). We generate 3,283 potential "squatting names" for these 10 apps, employing the rules of existing domain squatting tools. We then verify whether the potential squatting apps exist in the wild (using Koodous [20], a mobile app corpus containing more than 50 million Android apps). We find that app squatting abuse is, indeed, highly prevalent for Android apps. Based on the verified squatting apps, we then identify the common patterns that attackers leverage in app squatting (**Section 4**). We then use these rules to design and implement *AppCrazy*, a tool to systematically generate squatting identifier names for mobile apps. Given an Android app as the input, AppCrazy automatically generates confusingly similar app names and package names that could be leveraged by attackers. Exploiting the capabilities of AppCrazy, we then conduct a large scale empirical analysis: we apply it to 426 additional apps crawled from Google Play and generate more than 200K potential squatting names (**Section 5**). From these, we discover 10,553 squatting apps, confirming that this threat far exceeds the top 10 apps alone. Finally, we characterize the impact introduced by app squatting (**Section 6**), including the prevalence of squatting apps in major app markets, and the number of app installs of the squatting apps. To summarize, we make the following main contributions:

(1) We demonstrate that squatting attacks are prevalent in the mobile app ecosystem. To the best of our knowledge, this is the first comprehensive study of the characteristics and implications of *App Squatting* attacks.
(2) We identify 11 common patterns leveraged by app squatting attackers, and design *AppCrazy*, a tool to automatically generate squatting variations for Android apps.
(3) Using AppCrazy, we perform a *large-scale empirical study* to understand the squatting app phenomenon and investigate the lexical characteristics and malicious behaviors. We generate 224,332 potential squatting names for 426 apps, and discover 10,553 squatting apps. Worryingly, 51% are classified as malicious by VirusTotal.
(4) We study the *impact* of app squatting, discovering that squatting apps have been found in 33 app markets, including the official Google Play. Many squatting apps have gained a large number of app downloads (up to over 10 million).

We hope that our efforts can positively contribute to raise awareness among relevant stakeholders (including mobile users, app developers, and app market maintainers). Hence, we have open-sourced AppCrazy and the identified squatting apps at:

https://github.com/squattingapp/AppCrazy

2 BACKGROUND AND RELATED WORK

Domain Squatting. Domain squatting [9] is a well established attack vector. It is the act of registering a domain name very similar to an existing legitimate domain, in an effort to capture some of the (web) traffic going to the original domain. Domain squatting can be grouped into different categories based on different squatting techniques [101], e.g., typosquatting [30] (squatting via typographical errors), bit-squatting [96] (squatting via accidental random bit flips), homograph-based squatting [79, 83] (domains that abuse characters from different character sets), soundsquatting [95] (domains that abuse the pronunciation similarity of different words), and combo-squatting [88] (combination of a recognizable brand name with other keywords).

Typosquatting is the most popular squatting technique, and has been well studied by the research community. Wang *et al.* [117] proposed a general and widely adopted approach to generate typosquatting domain names. Given a target domain (e.g., www.facebook.com), various typo-generation models could be applied. The commonly used typo-generation models include "Missing-dot typos" (e.g., wwwfacebook.com), "Character-omission typos" (e.g., www.faceook.com), "Character-permutation typos" (e.g., www.faecbook.com), "Character-substitution typos" (e.g., www.facebooj.com), and "Character-duplication typos" (e.g., www.faceboook.com).

Whereas domain squatting is primarily associated with web-based attacks, there has been a recent spate of attacks applied to other areas. Szurdi *et al.* [103] studied email typosquatting. Several studies [1–4, 28] have also observed attacks in programming package managers, e.g., PyPI, RubyGems, and NPM (popular among developers of Python, Ruby, and JavaScript respectively). For example, malicious typosquatting libraries have been found on PyPI repository [2, 3, 28]. Tschacher [108] studied the feasibility of typosquatting attacks in package managers. Our work takes inspiration from these studies but differs fundamentally: *we explore the presence of squatting attacks within the mobile app ecosystem.*

Fake Apps and Repackaged Apps. Although many research efforts have been focused on security and privacy issues in the mobile app ecosystem [87, 91, 92, 110, 113, 114, 114, 119], prior work on squatting attacks in app stores is rather limited. There have been a number of studies on fake apps and repackaged apps (app clones). A "fake app" masquerades as the legitimate one by mimicking the look

or functionality [13]. As suggested by previous studies [112, 116], fake apps usually have identical app or package names to the original ones. While a "repackaged app" often shares a large portion of the code with the original app [74, 124] (e.g., by decompiling the original app and inserting a malicious payload), they are obviously signed by different developers.

Wang *et al.* [116] proposed a clustering approach on app names to detect potential fake apps. Tang *et al.* [105] collected over 150K fake apps that have same package names or app names with popular apps. Kywe *et al.* [89] proposed a technique to detect fake apps based on the external features of apps, e.g., icons, app names. Zhou *et al.* [87] found that more than 80% of malicious apps are distributed in the form of repackaged apps (with same package name). Besides, a number of studies proposed methods to detect repackaged apps based on simple hashing [82, 124], static semantic features [90, 94, 109, 123], resource signatures [99, 121], graph similarity [74–76] and UI birthmark [80, 100, 120], etc.

The focus of this paper differs from the above: whereas the previous work study fake apps, where the app identifiers are unaltered, we shed light on app squatting, where the attackers modify the app identifiers to trick users. Thus, we are agnostic to the implementation of the apps themselves and, instead, focus on how attackers strive to gain installations. To the best of our knowledge, there is no prior study on this topic despite significant media attention [14, 15, 35].

3 MOTIVATING STUDY

The aforementioned studies suggest that app squatting might be a serious problem. Accordingly, we perform a preliminary motivational study to (1) confirm the presence of squatting-like threats in the app ecosystem, and (2) test if existing domain squatting generation techniques correctly identify them. We do this to provide a ground truth that can help inform our later methodology design.

3.1 Methodology

To test for the presence of typosquatting we generate variations of several popular app and package names, and check whether they exist in app repositories.

Generating squatting names. We begin by selecting 10 popular apps from Google Play, each of which has over 100 million installations (see Table 1). For each app, we manipulate the app name and app ID (package name) to generate deformed names that could be leveraged by attackers to mislead users. Unfortunately, the number of deformed strings increases exponentially with the length of the original string. For example, if we take 36 characters (26 letters and 10 numbers) as substitutes, then a 5-letter word will generate thousands of deformed words (by changing just one or two characters).

To cope with this problem, we leverage existing efforts in generating squatting domains. State-of-the-art tools such as URLCrazy [31] and DNSTwist [10] are widely used for generating domain name typos to detect and perform domain squatting [78, 97, 107, 125]. These tools use similar generation models to produce deformed strings, covering most kinds of domain squatting attacks.

We perform this study using a representative tool, URLCrazy, which covers 15 generation models. As URLCrazy is specialized for domain squatting (i.e., the input must be a qualified domain name), we perform a number of steps before inputting app names to enable compatibility. We first add a top-level domain at the tail of the app name and package name, e.g., "com.facebook" is changed to "com.facebook.com". Further, as app names can contain a space (but a domain cannot), we replace any spaces with dots (e.g., "Youtube music" becomes "Youtube.music"). To increase the number of generated strings, we also input multiple orderings, (e.g., Youtube.music.com, music.Youtube.com). Using this approach, we create 3,283 deformed names (including 1,125 app name variations and 2,158 package name variants) for the 10 apps considered.

Verifying squatting names. To verify whether any of the newly generated names exist in the wild, we take advantage of Koodous, a collaborative platform focused on the detection of fraudulent patterns in Android apps [20], which is widely used in previous work [73, 81]. Koodous is an APK repository hosting more than *50 million* Android apps, by far the largest accessible Android app repository to the best of our knowledge.

We run an automated crawler to search Android apps on Koodous using the 3,283 generated names. This returns more than 2,136 results. We then filter these results to only leave those apps with exact string matches. This is necessary because Koodous returns many "related" apps that do not necessarily match our string query. Our filtering leaves 138 apks (125 package names and 98 app names). We then use search engines (including Google and Baidu) to remove false positives, by searching the app or package names directly to find whether the app is legitimate or not. This turned out to be a critical step, as we found 49 false positives. For example, for the social-networking app "Wechat", one of the generated app names by URLCrazy is "Wochat". However, this is another popular Android app [37] that should not be labelled as a squatting app.

3.2 Motivating Results

Through the above process we identified 89 squatting apps. We show detailed results in Column 2-4 of Table 1. Note that Table 1 also includes the results for our tool AppCrazy (later presented in Section 4). Although our approach is straightforward, we still manage to identify squatting apps for *all* 10 apps studied. For instance, we discover 28 squatting apps targeting Facebook; for context, Table 2 lists 5 app squatting examples. That said, for the 15 generation models used in URLCrazy, only 6 of them are shown to be effective in generating squatting apps. For the generated 3,283 name strings, only 26 of them are matched, which means that more than 99.2% of the generated strings are not effective in identifying squatting apps. Interestingly, through this process, we also encountered a number of squatting apps that were not identified using the strings generated by URLCrazy. For instance, as previously stated, Koodous returns "related" apps for each query; within these, we manually identified 827 further squatting apps (not listed in Table 1) that did not directly match the strings generated by URLCrazy.

3.3 Observations

The above study confirms our hypothesis that squatting attacks are prevalent in the mobile ecosystem. However, we find a number of limitations in domain generation approaches (i.e., URLCrazy):

Table 1: Results of the motivating study.

App Name (# of installations)	Domain Squatting Generation Models			AppCrazy		
	# of generated app names	# of generated package names	# of identified squatting apps	# of generated app names	# of generated package names	# of identified squatting apps
Instagram (**1B**) [18]	125	239	13	17	62	89
Pinterest (**100M**) [21]	133	156	1	22	32	16
Twitter (**500M**) [29]	108	252	7	16	55	97
Facebook (**1B**)[12]	109	231	28	30	67	205
Wechat (**100M**) [34]	88	156	6	12	39	310
Tumblr (**100M**) [27]	88	112	6	10	19	22
Telegram (**100M**) [24]	113	247	9	20	61	160
Youtube Music (**100M**) [38]	160	430	1	44	826	3
Tinder (**100M**) [25]	94	117	11	15	24	16
Snapchat (**500M**) [23]	107	218	7	16	55	28
Total	1125	2158	**89**	202	1240	**946**

Table 2: Example of five squatting apps targeting Facebook. We underline the difference with the legitimate app.

App Name	Package Name	Developer Name	APK_MD5
Facebook	com.facebook.katama	BombSoft	b4fcc28f58880f5592b63eac0c05e088
Facebooks	com.hdc.bookmark235646	truecaller	3f320887bba20d311c0fb399260c984c
facbook	net.droidjack.server	4PDA	601dfb76ba0f257103bcb78c4f4b3fc8
Facebookk	net.droidjack.sandrorat	Google	a9f0efd56cf4c1aaa6ae0f11e1a6d3cb
faceboook	my.app.client	Google	5d350a75d2938091066c8cb132ac140e

- Most of the generated variants did not identify *any* squatting-like apps. This creates a significant wasted overhead when testing for their presence in app stores.
- URLCrazy *misses* many squatting apps. As Koodous presents a list of related apps to the input keywords, we discovered a number of fake apps that did not match strings generated by URLCrazy. For example, some squatting apps targeting Facebook include those with the package name "com.facebook.litf", or with the app name "Facebook 2". These strings were not generated by URLCrazy.
- Existing generation models produce many *false positives*. Although in this motivating study we could manually remove false positives, this approach is not scalable.

Next, we seek to design a specific approach for generating and characterizing app squatting attacks. On the one hand, we must *improve existing domain name generation models to reduce inaccurate deformed strings* (Section 4). But, on the other hand, we must also *filter out false positive apps automatically* (Section 5).

4 APP SQUATTING-GENERATION MODELS

The straightforward approach demonstrates that squatting is indeed prevalent in the mobile app ecosystem, but that the existing tools are not adequate. This section presents *AppCrazy*, a tool for generating squatting app and package names with higher accuracy and recall (than existing tools designed for domain names).

4.1 Terminology

We define **App Squatting** as a type of squatting behavior where attackers release apps with identifiers that are confusingly similar to those belonging to popular apps or large Internet brands. Based on the target of the squatting, we classify apps into **app name squatting** and **package name squatting**. As mentioned in Section 2, fake apps have been widely studied; we differentiate squatting apps with traditional fake apps as follows:

(1) **Fake Apps:** Apps with an *identical* app or package name to legitimate apps, but with different developer signatures.
(2) **Squatting Apps:** Apps whose app or package name is confusingly *similar* (but *unidentical*) to the legitimate app.

Squatting-generation models are used to generate various deformed strings for a given input. We refer to the generated deformed strings as "**squatting names**".

4.2 Squatting Generation Models

AppCrazy consists of a set of models for generating potential squatting app names and package names. Here we present these models.

Critique of URLCrazy. Before we present the generation models for AppCrazy, we briefly revisit the limitations of existing models used for domain names. Although URLCrazy produces promising initial results, our motivating analysis identifies two problems:

Low Accuracy. URLCrazy generates many types of name variations that are *never* matched. Although URLCrazy provides 15 kinds of generation models for domain squatting, most of them are not suitable for mobile apps. For instance, "Bit Flipping" implements manipulation of binary digits, which will lead to large changes in the string appearance (e.g., "Facebook' into "nacebook"). As a result, most of the generated squatting names are ineffective.

Low Recall. URLCrazy ignores common patterns in app naming (because it is designed for generating domain names). During the search process on Koodous, we found many deformed strings not covered by URLCrazy. For example, many squatting names insert characters at the tail of the original string (e.g., Facebook 2 and Facebook Update), and some squatting package names partially replace the string (e.g., com.facebook.litf).

Our Generation Models. Thanks to our preliminary investigation, we have identified *11* squatting generation models for app identifiers. As shown in Figure 2, these models can be classified into two categories: (1) *mutation-based squatting generation models*, and (2) *combosquatting generation models*. To better illustrate the characteristics of the different models, the rest of this section examines the strings generated using these 11 techniques for Facebook (with package name *com.facebook.katana* as an example).

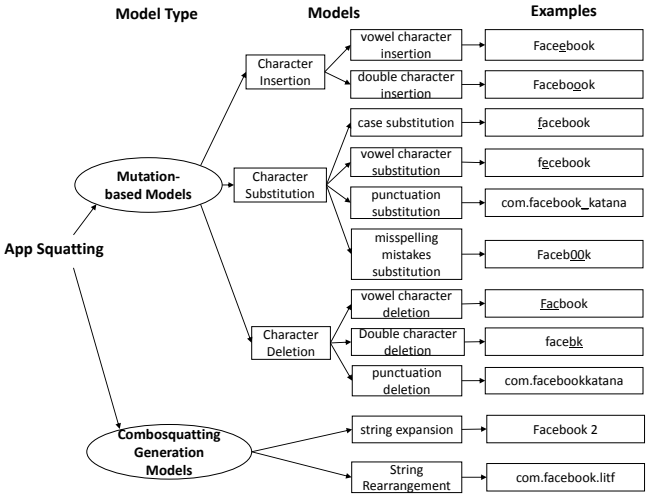


Figure 2: The 11 kinds of app squatting-generation models identified in this paper.

Mutation-based Models. These models generate squatting names based on either typographical errors or abusing the pronunciation similarity of different words. We now summarize 9 mutation-based models as follows:

(1) *Case Substitution*: Replacing an uppercase character with a lowercase one (or vice versa), e.g., "Facebook" into "facebook". Note that the package names in most app markets (e.g., Google Play) are case sensitive.
(2) *Vowel Character Insertion*: Inserting another vowel character after a vowel character, e.g., "Facebook" into "Faceebook".
(3) *Vowel Character Deletion*: Deleting one or more vowel characters, e.g., "Facebook" into "Facbook".
(4) *Vowel Character Substitution*: Replacing a vowel character with one of the other four vowel characters, e.g., "Facebook" into "Fecebook".
(5) *Double Character Insertion*: Inserting the same character between two consecutive identical characters, e.g., "Facebook" into "Faceboook".
(6) *Double Character Deletion*: Deleting one or two characters that are consecutively identical, e.g., "Facebook" into "Facebok", and "Facebook" into "Facebk".
(7) *Punctuation Substitution*: Replacing punctuation marks with other ones (including space, underscore and dot), e.g., "com.facebook.katana" into "com.facebook_katana".
(8) *Punctuation Deletion*: Deleting a punctuation mark (including space, underscore and dot), e.g., "com.facebook.katana" into "com.facebookkatana".
(9) *Common Misspelling Mistakes Substitution*: Replacing specific characters with 9 common misspelling mistakes[104]. e.g., "Facebook" into "Faceb00k".

Combosquatting Generation Models. These models rely on the combination of recognizable brand names with other keywords (as originally proposed in [88]). For instance, URLs such as paypal-members[.]com and facebookfriends[.]com could lead a user to believe the domains belong to PayPal and Facebook, respectively. In the case of apps, examples include "Paypal App 2", or "PayPal Update". App combosquatting differs from other forms of app squatting in two fundamental ways: (1) combosquatting does not involve any spelling deviation from the original app, and (2) it requires the original app identifier names to be intact within a set of other characters. As a result, we define two kinds of combosquatting generation models in this paper.

(1) *String Expansion*: Inserting characters before or after the identifier names, e.g., "Facebook" into "Facebook1".
(2) *String Rearrangement*: Splitting the string into elements based on the "dot" character, and rearrange the elements, e.g., "com.facebook.katana" into "com.katana.facebook" and "com.facebook". To improve accuracy, we discard rearranged strings that are composed of common names in Android (we collect 6 common strings).

We embed the above generation models in our tool, *AppCrazy*, which we have open-sourced. For an input of an app or package name, AppCrazy returns a list of potential squatting names.

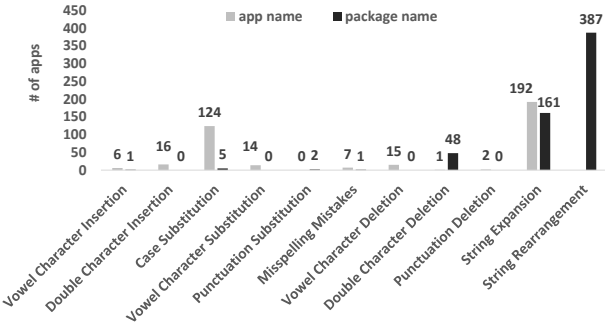


Figure 3: The distribution of squatting apps across models.

4.3 Evaluation of the Generation Models

Methodology. To evaluate the efficiency of the squatting generation models, we compare its results with the traditional domain squatting approaches used in the motivating study (see Section 3). We use the same set of 10 popular apps listed in Table 1. By feeding the 10 apps to AppCrazy, we generate 1,442 squatting names (202 deformed app names and 1240 deformed package names), as shown in Column 5-6 in Table 1.

We then proceed as in the motivating study by searching for the squatting names in Koodous. We download any returned apps (apks), and remove false positives (using search engines and app stores again). Note that for the combo squatting attacks (i.e., "String Expansion", "String Rearrangement"), we flag the apps as squatting candidates if their corresponding identifier names have an inclusion relationship with the input strings. This process identifies 5,315 squatting app candidates (with different MD5 hash value): 415 distinct app names and 872 distinct package names. After the manual removal of false positives, we identify 946 squatting apps.

Results. Roughly half of the squatting apps (452) correspond to the combo squatting attacks, and the other half (494) belong to mutation-based squatting attacks. Out of the 946 apps, 377 apps leverage app name squatting and 605 apps take advantage of package name squatting to mislead users. Figure 3 shows the number of squatting apps conforming to each of the the 11 generation models. All the models in AppCrazy are effective in detecting squatting-like

apps. Combosquatting is the most effective model, i.e., "String Expansion" and "String Rearrangment", with 353 and 387 squatting apps belonging to those categories, respectively.

The key advantages of AppCrazy:

- *Model Efficiency.* All of the 11 generation models proposed in AppCrazy are effective in detecting squatting apps, whereas only 6 out of 15 models in URLCrazy are successful.
- *Keyword Efficiency.* Out of 1,442 deformed strings generated by AppCrazy, 46 strings are effective in discovering squatting apps, whereas there are only 26 effective strings among the 3,283 deformed strings, which are generated by URLCrazy.
- *The number of identified squatting apps.* By applying AppCrazy to the same 10 apps, we could identify 946 squatting apps (apks), i.e., about 10 times more than using URLCrazy.

In summary, these initial experiments confirm that our tool is more effective in pinpointing squatting names than traditional domain squatting generation tools. In the following sections, we leverage AppCrazy to perform a large-scale measurement study of app squatting abuse in the wild (Section 5), then we characterize the impact of app squatting (Section 6).

5 MEASURING SQUATTING APPS

In this section, we exploit AppCrazy to broaden our analysis and perform a large-scale measurement of app squatting abuse in the wild. We therefore integrate AppCrazy into a measurement pipeline and collect a dataset covering attacks against 426 apps. Our measurement study is driven by the following research questions:

RQ1 ***What is the distribution of squatting apps in comparison with fake apps? What are the most popular generation techniques of app names?*** While fake apps have been widely studied, the presence of app squatting is not understood yet. We seek to investigate how widespread app squatting is, the differences with fake apps, and understand which squatting generation models are most popular.

RQ2 ***Does app squatting tend to target more popular apps?*** We seek to explore whether adversaries predominantly target apps with greater popularity.

RQ3 ***How many of the squatting apps are used for delivering malware?*** As previous work on domain squatting have suggested that phishing or spreading malicious contents is frequently the underlying motive [79, 83, 96], we seek to verify whether this threat is common in app squatting as well.

5.1 Methodology & Data Collection

To answer the above questions we integrate AppCrazy into a pipeline (see Figure 4) that: (1) Takes a series of app names as input, (2) Uses AppCrazy to generate a series of potential squatting names, (3) Queries Koodous to find matching apps, and (4) Filters false positives. We now describe these four steps and the collected dataset.

(1) Target App Selection. We first compile a list of legitimate app names, which may be subject to squatting attacks. As we mentioned earlier, while all mobile apps could be the subject of squatting abuse, it is arguably not in the best interest of an adversary to target a less known app. Consequently, we use the top 500 popular apps recommended by Google Play.[1] Note that Google Play recommends these apps based on app popularity in a given time slot, thus app downloads for these apps vary greatly, ranging from roughly 0.1 million (e.g., com.sports.real.golf.rival.online) to 1 billion (e.g., com.facebook.katana). We built this dataset in August 2018. Note that we discarded apps whose name contains non-English characters (e.g., "com.epi" uses a name with various non-English characters[2]) or specific non-standard characters (e.g., "com.konylabs.capitalone"[3] uses a name with ®). In the end, 426 apps are used.

(2) Keywords Generation & Searching. We next leverage AppCrazy to generate all the potential squatting strings for the app and package names. We then query Koodous with the squatting strings. Note that, to compare with the status of traditional fake apps, we also feed the original app and app package names to Koodous, in order to identify any fake apps that have identical app or package names with original ones.

(3) Model Matching. As mentioned, the search results returned by Koodous do not always match our input keywords accurately. Therefore we enforce a strict comparison to extract apps that have app or package names exactly matching our generation models. For mutation-based generation models, if the app or package name is identical to the generated strings, we flag the corresponding app as a squatting app candidate. For combosquatting generation models, we flag any apps that contain the name or package name (after the string rearrangement) of the target apps. For traditional fake apps, we flag any apps that have an app name or/and package name identical to the original ones as fake app candidates.

(4) App Filtering. We next discard any false positive squatting app candidates. Inspired by traditional domain squatting detection approaches [102], we propose a three-phase heuristic method to automatically remove false positives:

Same Developers. A developer might name the apps differently across platforms or devices. For example, the app "com.supercell.clashroyale" and app "com.supercell.clashroyale.samsung" are both legitimate apps (i.e., "Clash Royale"), released by the same developer but designed for different devices. Accordingly, we discard those squatting app candidates that having different names from the original apps, but are signed by the same developer.

Common Names. Some apps share widely-used names (e.g., Video Player [32] and Flashlight [16]), which are neither brand names (e.g., WhatsApp [36]) nor company names (e.g. Adobe [5]). This might introduce false positives. As a result, for an app whose name is composed of common English words, we consider it as a fake/squatting app only when its *package* name matches our predefined rules (i.e., we do not search for its app name). Our such words-list is collected from a public English common word list [8].

App/Developer White List. To further remove false positives, we create and use a reliable app/developer whitelist. To the best of our knowledge, no such datasets are available. Thus, we seek to build our own white list of apps and developers. We first rely on two datasets to collect the historical data of Google Play: (1) AndroZoo [7], a dataset with over 8 million apps and growing

[1] https://play.google.com/store/apps/top
[2] https://play.google.com/store/apps/details?id=com.epi
[3] https://play.google.com/store/apps/details?id=com.konylabs.capitalone

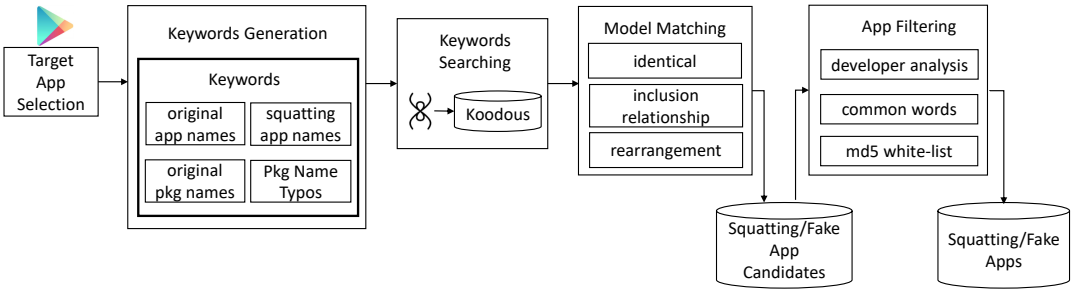


Figure 4: Our approach to identify squatting apps.

(mainly from Google Play), and (2) two historical snapshots of Google Play created in 2015 (over 1.5 million app) and 2017 (over 2.1 million app) by Wang *et al.* [111, 112, 115]. Additionally, we crawl a snapshot of Google Play ourselves in January 2019 (over 2.1 million apps). Using this data, we then compare the historical snapshots with the one we recently crawled, and *populate the white list with any app/developer that has not been removed by Google for over 1 year*. Although malware and fake apps are recurrently found on Google Play, they are removed once discovered. We later confirm that Google is extremely effective at removing these apps, validating this assumption (Table 8). By the time of our study, we have created a list of over 743K apps and 230K developer signatures.

Results. Using the pipeline, we have generated 224,322 squatting names for 426 target apps. We have fed these strings into Koodous to identify 106,316 squatting/fake app (apk) candidates that matched our rules. Our three-phase heuristic approach discarded 74,273 false positives, with each phase filtering about 2K, 30K and 40K apps, respectively. This left *32,043* fake or squatting apks. This confirm that fake/squatting apps are prevalent for the 426 target apps.

Accuracy. We randomly sample 10 target apps from the 426 apps for verification. These 10 apps have been targeted by 980 squatting apps and 1,242 fake apps in total. Although we have adapted the widely used approach (in domain squatting) of white listing to filter out false positives automatically, the list may be incomplete. To confirm whether the sampled apps in our dataset are truly fake/squatting apps, the first two authors verify these apps manually based on search engines to see whether the apps are legitimate or not. For the apps with disagreement or we cannot make sure, we label them as "potentially false positives". Eventually, we identify 9 false positives (0.9%) and 109 potentially false positives (9%) from the 980 squatting apps, and 66 potentially false positives (5%) from the 1,242 fake apps. This result suggests that, our approach with the automated app filtering can achieve at least 90% of accuracy.

5.2 RQ1: Distribution of Squatting Apps

We first investigate the distribution of squatting apps in the wild compared to fake apps, and the most popular app squatting patterns.

Fake vs. Squatting. For the 32,043 apps flagged, 27,103 are fake apps that have at least one identifier name (app name or package name) matching the original app. 10,553 match our squatting generation model, i.e., using confusingly similar identifier names. Note that an app could match both the rules of fake *and* squatting apps (e.g., using an identical package name and a confusingly similar app name). The large number of squatting apps found suggests that *squatting apps are rife and should be a cause of concern.*

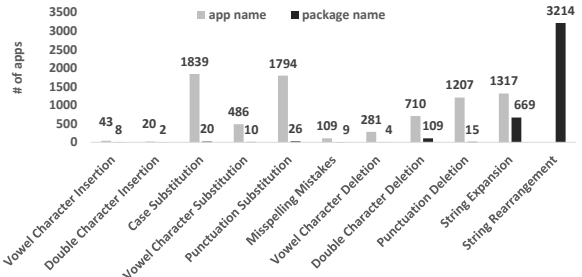


Figure 5: Distribution of squatting generation models.

Table 3: Legitimate apps targeted by fake and squatting apps.

	# apps (%) targeted by at least		
	1	10	100
fake apps	343(80%)	**216(51%)**	66(15%)
squatting apps	274(64%)	**106(25%)**	20(5%)
Total	375(88%)	**236(55%)**	69(16%)

Table 3 presents the distribution. Out of these 426 target apps, 343 of them have at least one fake app and 274 apps have at least one squatting app targeting them. Specifically, 51% of them (216 apps) have more than 10 fake apps and 15% of them (66 apps) have more than 100. 25% of the apps (106 apps) have more than 10 squatting apps and 5% apps (20) have more than 100 squatting apps. Table 6 lists the top 5 apps with the most fake and squatting apps (over 1000 per app). Note that three of them are game apps and two of them are social apps, which suggests these are key targets.

Most Popular Squatting Patterns. Figure 5 shows the distribution of the 11 proposed matching models in the collected 10,553 squatting apps. We make three key observations. First, these apps tend to rely more on app name squatting (6,306 apps) than package name squatting (3,936 apps). This is intuitive as typical users are likely to only inspect the app name when selecting apps. Second, for app name squatting, adversaries are more likely to modify the app name using "Case Substitution", "String Expansion" and "Punctuation Substitution". Third, for package name squatting, adversaries generally rely on the pattern of "String Rearrangement", e.g, we found a squatting app with package name "com.android.twitter", which is originated from "com.twitter.android". This can add insight for markets seeking to identify offending apps.

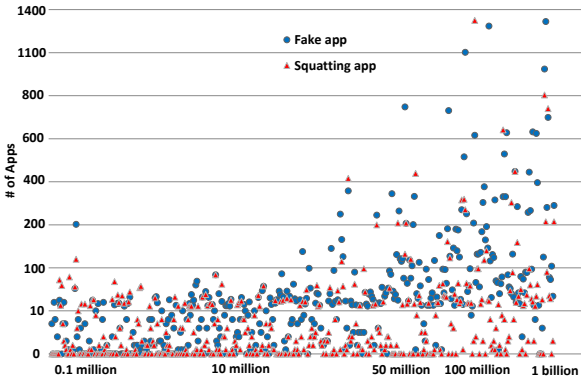


Figure 6: The distribution of app downloads vs. the number of fake (blue)/squatting (red) apps per target app.

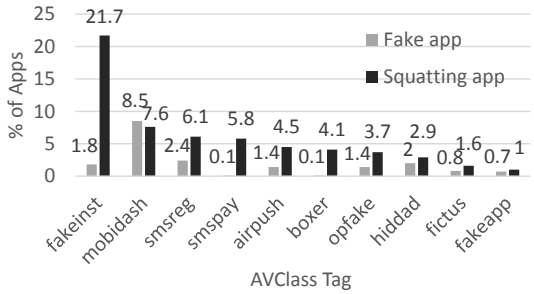


Figure 7: Top 10 AVClass families of squatting apps.

Table 4: Distribution of potential malware.

	Not-legitimate # apps (%) with AV-rank		
	≥ 1	≥ 10	≥ 20
Fake apps	21371(79%)	**5781(21%)**	3686(14%)
Squatting apps	7919(79%)	**5736(57%)**	4716(47%)
Total	25525(80%)	**9892(31%)**	7312(23%)

5.3 RQ2: App squatting vs. App Popularity

We next ask if attackers target more popular apps. We first rank the 426 apps by the number of downloads and compare the distribution of fake and squatting apps across these 426 apps (see Figure 6). Fake and squatting apps are prevalent for all the apps we studied, although apps with more downloads are likely to have more fake and squatting apps. Roughly 83% of fake apps (22,495 apps) and 76% of squatting apps (8,020 apps) are targeting the top third of apps (141 apps) who have over 50 million downloads. Roughly 22% of fake apps (5,963 apps) and 18% of squatting apps (1,902 apps) are targeting the top 5% of apps (22 apps) whose downloads is higher than 500 million. This intuitive finding indicates that adversaries do, indeed, focus on popular apps when performing squatting attacks.

5.4 RQ3: Malware Presence

Finally, we examine how many of the squatting apps are used for delivering malware. Hence, we uploaded all the apps to VirusTotal [33], a frequently used online analysis service that aggregates more than 60 anti-virus engines. As previous studies [71, 118] have suggested that some anti-virus engines may not always report reliable results, we analyze the results grouped by how many engines (AV-rank) flag an app as malware. Previous work [86, 122] has suggested 10 engines as a robust threshold.

Malware Presence. Table 4 shows that around 80% of the collected apps are flagged by at least one anti-virus engine. When using the threshold of "AV-Rank>=10", around 31% of the apps are labelled as malware. However, the malicious behavior of fake and squatting apps is remarkably different. More than 57% of the squatting apps are identified as malware with "AV-Rank>=10", while the percentage for fake apps is only 21%. This result suggests that squatting apps tend to be more malicious than fake apps. Table 5 lists the top 5 squatting-based malware according to their AV-Ranks. For example, the app "com.software.android.install" was flagged by 45 engines ("fakeinst" family), which is actually a Trojan appearing as installers for other apps [26].

Malware Category. We next analyze the distribution of malware categories and families reported by VirusTotal engines. The malware signatures for these apps mainly correspond to 5 general categories of malware: PUP/PUA (52.5%), Trojan (34.4%), Malware (27.4%), Adware (17.1%) and Riskware (14%). We also use AVClass [98], a widely used malware labeling tool to obtain the family name of each identified malware. Figure 7 presents the top 10 malware family of squatting apps, Families "fakeinst" [26] and "mobidash" [6] are the most popular: more than 21.7% (2290 squatting apps) and 7.6% (802 squatting apps) of flagged malicious apps belong to them. Furthermore, the distributions of malware families differ greatly between fake and squatting apps, confirming again that these two types of malicious app are distinct.

6 CHARACTERIZING THE IMPACT

We now further characterize the wider impact of app squatting by exploring the following research questions:

RQ4 **How prevalent is the problem in major Android app distribution channels?** Although Koodous is by far the largest Android app repository, the apps collected may come from different app distribution platforms that are not explicitly identified in Koodous (there is no source information). Thus, it is interesting to further explore the presence of squatting apps in major markets.

RQ5 **How many users would be tricked into installing these apps?** Although we have identified a large number of squatting apps, the impact it causes to the wider mobile app ecosystem is still unknown. One of the most explicit ways to measure this is the number of app downloads.

6.1 Methodology & Data Collection

To answer the aforementioned RQs, we first harvest a dataset with app market information. We leverage three up-to-date and large-scale app datasets to identify squatting apps in major app markets:

- **Dataset from Wang *et al.* [116]**, created in August 2017 and with over 6.2 million app items collected from Google Play and the 17 most popular Chinese app markets.
- **AndroZoo Dataset [7]**, an academic effort focused on compiling a large-scale dataset of APKs. We use the dataset from March 2019. It contains more than 8.8 million apks from 16 app markets. Roughly 80% are from Google Play.

Table 5: Top 5 squatting-based malicious apps according to AV-Rank.

App Name	Package Name	MD5	AV-Rank	malware family
Angry_Birds	com.software.android.install	81ab22a9700f3db8f3a22cb6544a9166	45	fakeinst
Angry_Birds	com.software.application	2258bd0efa20e732fa7df1442d6f3f08	45	boxer
Angry_Birds	com.software.app	465f95ef5e703a76722252a377ac10ce	45	fakeinst
Spider Man	powerstudio.spiderman	f770fbf102d1d980515c8a6af4ef5c24	44	lotoor
Angry_Birds	com.googleapps.ru	0317519f2f2d7b11eba9edeff45f2967	44	fakeinst

Table 6: Top 5 apps with the most number of fake and squatting apps.

App name	Package name	# of Fake Apps	# of Squatting Apps	# of App Name Squatting Apps	# of Package Name Squatting Apps	Total
Angry Birds	com.rovio.angrybirds	616	1314	1313	26	1928
Skype	com.skype.raider	1284	164	22	148	1376
Clash of Clans	com.supercell.clashofclans	1113	270	255	15	1362
8 Ball Pool	com.miniclip.eightballpool	1296	23	22	1	1319
WhatsApp	com.whatsapp	996	814	221	524	1162

Table 7: Coverage of the App Store Datasets.

Dataset	Period	# Apps	# App Stores
Wang et al. [116]	2017	5.6M	18
AndroZoo [7]	2016-2019	8.8M	16
Janus [19]	2014-2019	15.2M	23

- **Janus Dataset [19],** a growing mobile security data corpus containing more than 15 million Android apps. The apps in the Janus dataset were explicitly labeled with their source markets (over 22 app markets).

We list the details of these three dataset in Table 7. Overall, we cover 33 different app markets.

6.2 RQ4: Presence of Squatting in App Markets

We first investigate how prevalent the problem is in major Android app distribution channels. We repeat the process introduced in Section 5 to check the presence of squatting apps in the above three datasets. The only difference is that we replace the app corpus (Koodous) with the three app market datasets mentioned above. This is because the three datasets have the added benefit of including metadata about which market the app is hosted in. Table 8 summarizes the number of squatting apps across the 33 markets. There are squatting apps in *all* markets. We find 1,794 apps with 3,930 different versions across the markets. 1,500 squatting apps (with 897 distinct package names) have even been published in the official Google Play. Besides Google Play, third-party app markets are the main distribution channels for the spreading squatting apps.

Whereas the largest portion of squatting apps can only be found in a single market, we observe that the majority are actually replicated across *multiple* markets. For context, we list the top 3 apps with the largest number of squatting attacks across the markets in Table 9. Each of these heavily targeted apps have hundreds of squatting apps in at least 20 markets. From the 426 popular target apps, 55% of them (236 apps) have suffered squatting attacks in at least one market. For example, the squatting apps of Angry Birds have been published on 29 different markets. We have identified various aggressive cases. For example, "Faceboook" with the squatting package name "com.bryan.faceboook"[4] was found in Google Play, MyApp, Wandoujia, AnzhuoApk, OPPO and 25PP. "WhatsApp" with the squatting package name "com.gbwhatsapp"[5] was

Table 8: Distribution of Squatting Apps across app stores. In each dataset column we list the number of apps with unique package names, and the number of distinct APKs with unique MD5 hashes (in parentheses).

App Store	Wang et al. (APK/APP)	AndroZoo (APK/APP)	Janus (APK/APP)	Total
EOE Market [52]	–	–	322(242)	322(242)
25PP [40]	810(198)	–	368(269)	1153(450)
Baidu Marjet [48]	419(66)	–	1191(768)	1449(810)
Coolpad [50]	–	–	133(110)	133(110)
Huawei Market [55]	154(8)	–	234(176)	447(207)
Flyme Market [58]	310(33)	–	252(189)	553(215)
Appchina [47]	270(16)	428(74)	1106(718)	1497(774)
AnzhuoApk [45]	–	–	983(646)	983(646)
360 Market [41]	258(47)	–	727(485)	975(525)
Google Play [17]	125(57)	799(469)	736(163)	1500(897)
Apkpure [46]	–	–	290(218)	290(218)
Wandoujia [68]	700(124)	–	648(432)	1317(540)
Liqu [57]	344(38)	–	116(97)	458(133)
Uptodown [67]	–	–	319(232)	319(232)
ZOL [70]	144(8)	–	58(53)	202(61)
MyApp Market [62]	467(104)	–	319(225)	780(323)
Lenovo Market [56]	237(14)	–	100(79)	334(90)
Xiaomi Market [69]	284(20)	218(13)	79(54)	372(76)
SouGou Market [66]	401(42)	–	187(143)	582(179)
Mumayi [61]	–	–	74(63)	74(63)
Anzhi [44]	339(44)	624(102)	163(116)	779(211)
Mobiseclab [60]	–	–	50(47)	50(47)
Angeeks [43]	–	253(27)	–	253(27)
1mobile [39]	–	27(13)	–	27(13)
Freeware Lovers [53]	–	37(3)	–	37(3)
91 Assistant [42]	165(8)	–	–	165(8)
Dangle [51]	88(21)	–	–	88(21)
OPPO Market [64]	369(73)	–	–	369(73)
PCOnline [65]	257(39)	–	–	257(39)
NDuo [63]	93(5)	–	–	93(5)
MGYApp [59]	122(17)	–	–	122(17)
GFan [54]	60(5)	–	–	60(5)
CNMO [49]	120(7)	–	–	120(7)
Others	–	–	665(447)	665(447)
Total	1162(342)	1869(617)	2291(1305)	3930(1794)

[4]MD5:0a69fa5aa4d99328d49a99aa327f0023

[5]MD5:8c3281591f99d81615edac56e8f04ea2

Table 9: Top 3 targeted apps according to the number of squatting apps across stores.

App Name	Package Name	# Sqt Apps	# Stores
Angry Birds	com.rovio.angrybirds	557	28
Spider-Man	com.gameloft.android.ANMP.GloftSIHM	248	29
Youtube	com.google.android.youtube	232	24

found in Google Play, Baidu, AnzhuoApk, MyApp, Sougou, Huawei, CoolPad, Meizu, 25App, Mumayi, Uptodown and Apkpure. This confirms that some attackers extensively replicate their squatting apps across many markets to gain greater visibility. This also suggests that the *markets do not perform name or app typosquatting checks, and could potentially benefit from sharing information.*

6.3 RQ5: Impact on App Downloads

We now move our attention to the impact of squatting apps. In particular we wish to understand how successful these squatting apps are in tricking users into installing them. Note that for the three datasets used, only [116] contains the number of app installs. Thus, we further refer to this dataset where we have identified 342 squatting apps with unique package names (1,162 apk versions), which target 67 legitimate apps. Note that for the remaining 359 legitimate apps, we did not identify their corresponding squatting apps in this dataset. Table 8 summarizes these results.

Downloads of Squatting Apps. We first analyze the distribution of app downloads for all the 342 identified squatting apps. 34% of them (117 apps) have over 1K downloads, and 8% of them (28 apps) have more than 100K downloads, with the largest one reaching over 10 million. Overall, they have been downloaded 59 million times. *This result confirms the efficacy of the app squatting attacks.*

Downloads of Legitimate vs. Squatting app. We further study how popular the squatting apps are in comparison to the legitimate ones. For 12 out of the 67 legitimate apps, over 1% of the downloads corresponds to the squatting versions rather than the legitimate app. Interestingly, for a subset of 8 apps, the number of squatting app downloads goes beyond 10% of the downloads of the original apps. We also identify one extreme case (app "com.squareup.cash") where the number of downloads of the squatting version equals those of the legitimate app. *These results suggest that squatting apps do have a meaningful impact on the original apps, which might lose potential users, and suffer a negative impact on their brand.*

7 DISCUSSION

7.1 Mitigation & Implications

We identify a number of methods to mitigate the severity of app squatting; we discuss them with three stakeholders in mind.

App Market Maintainers. We argue that there is a need to design policies to regulate app naming schemes. For example, if a given app named *Telegram* exists already in the market, other apps using similar names (such as *Teligram*) should be disallowed, or extensively scrutinized. AppCrazy can assist in this process. Equally, if policy violating apps are found in search results (e.g., Bing), the search engine should highlight such violations (and report them to the market).

App Developers. App developers should also be made aware of squatting abuse so as to invent a name that is not similar to other (existing ones). They should also take the responsibility to search for and identify squatting that target their apps. In such cases, developers could then take actions to mitigate possible abuses (e.g., by reporting them to the market maintainers).

Mobile Users. Awareness should also be raised among users. For educational purposes, we commit to post regular tutorials and reports on our website to provide a means for market maintainers, app developers as well as app users to learn more about app squatting attacks. We have also released our tool to the community, and will further introduce an online web service that takes as input an app (or package) name and outputs a list of name variants that could be leveraged to check for app squatting attacks. The web service will also illustrate if the given name variant has actually been adopted by an existing app.

7.2 Limitations

Our study carries certain limitations. First, we have identified 11 squatting generation models for app identifiers based on our motivating study (implemented within AppCrazy). Although far more accurate than URLCrazy, it is still possible that the generation models are incomplete, and other sophisticated methods exist. To alleviate this, we have designed AppCrazy as an easy-to-extend tool, where new patterns can be readily added. Second, when filtering false positives, we have adapted the widely used approach (in domain squatting) of white listing. Accordingly we have created and released a large white list of apps and developer signatures to filter false positives. While this proved effective, we acknowledge that the list may be incomplete. To the best of our knowledge, it is non-trivial to discard false positives (both for URL and app squatting) and we could identify no better alternatives. Finally, we also note that our study has primarily focused on popular apps. We argue this is appropriate as we found that attackers primarily target these well known apps. Despite this, our future work will explore how our findings generalize across the full popularity spectrum.

8 CONCLUDING REMARKS

This paper has presented the first in-depth measurement study of app squatting attacks. Our study has revealed that squatting attacks are prevalent in the mobile app ecosystem, thereby motivating the need for more efforts to identify and prevent potential abuses. We have identified common patterns that adversaries leverage to perform app squatting attacks, and developed a tool (AppCrazy) for the automatic generation of squatting names. By applying AppCrazy to 426 popular apps, we have discovered more than 10K squatting apps, many of which are used for delivering malware.

ACKNOWLEDGMENT

This work was partly supported by the National Key Research and Development Program of China (No. 2018YFB0803603), by the National Natural Science Foundation of China (No. 61702045 and No. 61772042), by the Australian Research Council (ARC) under projects DE200100016 and DP200100020, by the Alan Turing Institute (EPSRC EP/N510129/1) and grant EP/P025374/1.

REFERENCES

[1] 2017. Attackers Use Typo-Squatting To Steal npm Credentials. https://threatpost.com/attackers-use-typo-squatting-to-steal-npm-credentials/127235/.
[2] 2017. PyPI Python repository hit by typosquatting sneak attack. https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack/.
[3] 2017. Ten Malicious Libraries Found on PyPI - Python Package Index. https://www.bleepingcomputer.com/news/security/ten-malicious-libraries-found-on-pypi-python-package-index/.
[4] 2017. This typosquatting attack on npm went undetected for 2 weeks. https://www.theregister.co.uk/2017/08/02/typosquatting_npm/.
[5] 2018. Adobe Scan. ttps://play.google.com/store/apps/details?id=com.adobe.scan.android.
[6] 2018. Android/Adware.MobiDash. https://blog.malwarebytes.com/detections/android-adware-mobidash.
[7] 2018. Androzoo. https://androzoo.uni.lu/.
[8] 2018. Common English Words. https://github.com/first20hours/google-10000-english.
[9] 2018. Cybersquatting - Wikipedia. https://en.wikipedia.org/wiki/Cybersquatting.
[10] 2018. DNSTwist: domain name permutation engine. https://github.com/elceef/dnstwist/.
[11] 2018. Dot-cm Typosquatting. ttps://krebsonsecurity.com/2018/04/dot-cm-typosquatting-sites-visited-12m-times-so-far-in-2018.
[12] 2018. Facebook. https://play.google.com/store/apps/details?id=com.facebook.katana.
[13] 2018. Fake mobile apps, a growing threat. https://www.guardsquare.com/en/blog/fake-mobile-apps-growing-threat.
[14] 2018. Fake Teleg'e'ram on Google Play. https://www.zscaler.com/blogs/research/fake-telegeram-google-play.
[15] 2018. Fake WhatsApp app fooled million Android users on Google Play: Did you fall for it? https://www.zdnet.com/article/fake-whatsapp-app-fooled-million-android-users-on-google-play-did-you-fall-for-it/.
[16] 2018. Flashlight. https://play.google.com/store/apps/details?id=app.real.flashlight.
[17] 2018. GooglePlay. https://play.google.com.
[18] 2018. Instagram. https://play.google.com/store/apps/details?id=com.instagram.android.
[19] 2018. Janus. https://www.appscan.io.
[20] 2018. Koodous. https://koodous.com.
[21] 2018. Pinterest. https://play.google.com/store/apps/details?id=com.pinterest.
[22] 2018. PyPI Python repository hit by typosquatting sneak attack. https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack.
[23] 2018. Snapchat. https://play.google.com/store/apps/details?id=com.snapchat.android.
[24] 2018. Telegram. https://play.google.com/store/apps/details?id=org.telegram.messenger.
[25] 2018. Tinder. https://play.google.com/store/apps/details?id=com.tinder.
[26] 2018. Trojan:Android/Fakeinst. https://www.f-secure.com/v-descs/trojan_android_fakeinst.shtml.
[27] 2018. Tumblr. https://play.google.com/store/apps/details?id=com.tumblr.
[28] 2018. Twelve malicious Python libraries found and removed from PyPI. https://www.zdnet.com/article/twelve-malicious-python-libraries-found-and-removed-from-pypi/.
[29] 2018. twitter. https://play.google.com/store/apps/details?id=com.twitter.android.
[30] 2018. Typosquatting - Wikipedia. https://en.wikipedia.org/wiki/Typosquatting.
[31] 2018. URLCrazy. https://www.morningstarsecurity.com/research/urlcrazy.
[32] 2018. Video Player. https://play.google.com/store/apps/details?id=com.enhance.videoplayer.free.
[33] 2018. VirusTotal. https://www.virustotal.com.
[34] 2018. WeChat. https://play.google.com/store/apps/details?id=com.tencent.mm.
[35] 2018. What is Teligram? Fake Telegram app found serving up malware and ads on Google Play Store. https://www.ibtimes.co.uk/what-teligram-fake-telegram-app-found-serving-malware-ads-google-play-store-1655019.
[36] 2018. Whatsapp. https://play.google.com/store/apps/details?id=com.whatsapp.
[37] 2018. Wochat. https://play.google.com/store/apps/details?id=io.wochat.app.
[38] 2018. YouTube Music. https://play.google.com/store/apps/details?id=com.google.android.youtube.
[39] 2019. 1Mobile. http://www.1mobile.com.
[40] 2019. 25PP Market. https://www.25pp.com/android.
[41] 2019. 360 Market. http://zhushou.360.cn/?_gtype=guagua.
[42] 2019. 91 Assistant. http://zs.91.com/resourcea-app.html?page=1&type=android&restype=soft.
[43] 2019. Angeeks. http://apk.angeeks.com.
[44] 2019. Anzhi Market. http://www.anzhi.com.
[45] 2019. AnzhuoApk. http://www.anzhuoapk.com.
[46] 2019. Apkpure. https://apkpure.com/cn.
[47] 2019. AppChina. http://www.appchina.com.
[48] 2019. Baidu Market. https://shouji.baidu.com.
[49] 2019. CNMO. http://app.cnmo.com/android.
[50] 2019. Coolpad Market. https://www.coolmart.net.cn.
[51] 2019. Dangle Market. https://app.d.cn.
[52] 2019. EOE Market. http://www.eoemarket.com.
[53] 2019. Freeware Lovers. http://www.freewarelovers.com/apps.
[54] 2019. GFan. http://apk.gfan.com/gamess_8_1_1.html.
[55] 2019. Huawei Market. http://app.hicloud.com.
[56] 2019. Lenovo Market. https://www.lenovomm.com.
[57] 2019. Liqu Market. https://www.liqucn.com.
[58] 2019. Meizu Market. http://app.flyme.cn/apps/public/index.
[59] 2019. MGYApp. http://www.mgyapp.com.
[60] 2019. Mobiseclab. http://akana.mobiseclab.org.
[61] 2019. Mumayi. http://www.mumayi.com.
[62] 2019. MyApp Market. https://sj.qq.com/myapp.
[63] 2019. NDuo. http://www.nduo.cn/Home/Index/0/?webType=web.
[64] 2019. OPPO Market. https://store.oppomobile.com.
[65] 2019. PCOnline. https://dl.pconline.com.cn/android.
[66] 2019. SouGou Market. http://zhushou.sogou.com/apps.
[67] 2019. Uptodown. https://www.uptodown.cc.
[68] 2019. Wandoujia. https://www.wandoujia.com.
[69] 2019. Xiaomi Market. http://app.mi.com.
[70] 2019. ZOL Market. http://sj.zol.com.cn.
[71] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Ndss*, Vol. 14. 23–26.
[72] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. 2011. Measuring pay-per-install: the commoditization of malware distribution.. In *Usenix Security Symposium*. 13–13.
[73] Fabio CHAKRABORTY, Tanmoy; PIERAZZI. 2017. Ensemble clustering and classification for predicting android malware families. In *IEEE Transactions on Dependable and Secure Computing*. IEEE, 1–1.
[74] Peng Liu Chen, Kai and Yingjun Zhang. 2014. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *The 36th International Conference on Software Engineering*. ACM, 175–186.
[75] Jonathan Crussell, Clint Gibler, and Hao Chen. 2012. Attack of the clones: Detecting cloned applications on android markets. In *European Symposium on Research in Computer Security*. Springer, 37–54.
[76] Jonathan Crussell, Clint Gibler, and Hao Chen. 2013. Scalable semantics-based detection of similar android applications. In *Proc. of ESORICS*, Vol. 13. Citeseer.
[77] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. 2018. 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 (ESEC/FSE '18)*. 257–268.
[78] Yahia Elsayed and Ahmed Shosha. 2018. Large scale detection of IDN domain name masquerading. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 1–11.
[79] Evgeniy Gabrilovich and Alex Gontmakher. 2002. The homograph attack. *Commun. ACM* 45, 2 (2002), 128.
[80] Olga Gadyatskaya, Andra-Lidia Lezza, and Yury Zhauniarovich. 2016. Evaluation of Resource-Based App Repackaging Detection in Android. In *Nordic Conference on Secure IT Systems*. Springer, 135–151.
[81] Ali GHARIB, Amirhossein; GHORBANI. 2017. DNA-Droid: a real-time android ransomware detection framework. In *International Conference on Network and System Security*. Springer, 184–198.
[82] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. 2012. Juxtapp: A scalable system for detecting code reuse among android applications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 62–81.
[83] Tobias Holgers, David E Watson, and Steven D Gribble. 2006. Cutting through the Confusion: A Measurement Study of Homograph Attacks. In *USENIX Annual Technical Conference*. 261–266.
[84] Yangyu Hu, Haoyu Wang, Li Li, Yao Guo, Guoai Xu, and Ren He. 2019. 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, 332–343.
[85] Yangyu Hu, Haoyu Wang, Yajin Zhou, Yao Guo, Li Li, Bingxuan Luo, and Fangren Xu. 2019. Dating with scambots: Understanding the ecosystem of fraudulent dating applications. *IEEE Transactions on Dependable and Secure Computing* (2019).
[86] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kâafar, Noha Loizon, and Roya Ensafi. 2019. The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading. *Web Conference* (2019).
[87] Xuxian Jiang and Yajin Zhou. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*. IEEE,

95–109.

[88] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gomez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. 2017. Hiding in plain sight: a longitudinal study of combosquatting abuse. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 569–586.

[89] Su Mon Kywe, Yingjiu Li, Robert H Deng, and Jason Hong. 2014. Detecting camouflaged applications on mobile application markets. In *International Conference on Information Security and Cryptology*. Springer, 241–254.

[90] Li Li, Tegawendé F Bissyandé, Haoyu Wang, and Jacques Klein. 2019. On identifying and explaining similarities in android apps. *Journal of Computer Science and Technology* 34, 2 (2019), 437–455.

[91] Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM conference on ubiquitous computing*. 501–510.

[92] Tianming Liu, Haoyu Wang, Li Li, Guangdong Bai, Yao Guo, and Guoai Xu. 2019. DaPanda: Detecting Aggressive Push Notifications in Android Apps. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 66–78.

[93] Tianming Liu, Haoyu Wang, Li Li, Xiapu Luo, Feng Dong, Yao Guo, Liu Wang, Tegawendé F Bissyandé, and Jacques Klein. 2020. MadDroid: Characterising and Detecting Devious Ad Content for Android Apps. In *The World Wide Web Conference*. 1988–1999.

[94] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE-C '16)*. 653–656.

[95] Nick Nikiforakis, Marco Balduzzi, Lieven Desmet, Frank Piessens, and Wouter Joosen. 2014. Soundsquatting: Uncovering the use of homophones in domain squatting. In *International Conference on Information Security*. Springer, 291–308.

[96] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. 2013. Bitsquatting: Exploiting bit-flips for fun, or profit?. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 989–998.

[97] Yuta Sawabe, Daiki Daiki, Mitsuaki Akiyama, and Shigeki Goto. 2018. Detecting Homograph IDNs Using OCR. *Proceedings of the Asia-Pacific Advanced Network* 46 (2018), 56–64.

[98] et al Sebastián, Marcos. 2016. Avclass: A tool for massive malware labeling.. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer.

[99] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. 2014. Towards a scalable resource-driven approach for detecting repackaged android applications. In *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 56–65.

[100] Charlie Soh, Hee Beng Kuan Tan, Yauhen Leanidavich Arnatovich, and Lipo Wang. 2015. Detecting clones in android applications through analyzing user interfaces. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 163–173.

[101] Jeffrey Spaulding, Shambhu Upadhyaya, and Aziz Mohaisen. 2016. The landscape of domain name typosquatting: Techniques and countermeasures. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 284–289.

[102] Janos Szurdi, Kocso Balazs, Cseh Gabor, Spring Jonathan, Felegyhazi Mark, and Kanich Chris. 2014. The long "taile" of typosquatting domain names. In *Usenix Security Symposium*. 191–206.

[103] Janos Szurdi and Nicolas Christin. 2017. Email Typosquatting. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. 419–431.

[104] Bohm T. 2014. Letter and symbol misrecognition in highly legible typefaces for general, children, dyslexic, visually impaired and ageing readers. In *Information Design Journal*. 34–50.

[105] Chongbin Tang, Sen Chen, Lingling Fan, Lihua Xu, Yang Liu, Zhushou Tang, and Liang Dou. 2019. A large-scale empirical study on industrial fake apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 183–192.

[106] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. 2016. Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software.. In *USENIX Security Symposium*. 721–739.

[107] Ke Tian, Steve TK Jan, Hang Hu, Danfeng Yao, and Gang Wang. 2018. Needle in a haystack: tracking down elite phishing domains in the wild. In *Proceedings of the Internet Measurement Conference (IMC '18)*. ACM, 429–442.

[108] Nikolai Philipp Tschacher. 2016. *Typosquatting in programming language package managers*. Ph.D. Dissertation. Universität Hamburg, Fachbereich Informatik.

[109] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. 2015. 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*. ACM, 71–82.

[110] Haoyu Wang, Jason Hong, and Yao Guo. 2015. 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*. 1107–1118.

[111] Haoyu Wang, Hao Li, and Yao Guo. 2019. Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play. In *The World Wide Web Conference*. 1988–1999.

[112] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. 2018. 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, 231–242.

[113] Haoyu Wang, Yuanchun Li, Yao Guo, Yuvraj Agarwal, and Jason I Hong. 2017. Understanding the purpose of permission use in mobile apps. *ACM Transactions on Information Systems (TOIS)* 35, 4 (2017), 1–40.

[114] Haoyu Wang, Hongxuan Liu, Xusheng Xiao, Guozhu Meng, and Yao Guo. 2019. Characterizing Android App Signing Issues. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 280–292.

[115] Haoyu Wang, Zhe Liu, Yao Guo, Xiangqun Chen, Miao Zhang, Guoai Xu, and Jason Hong. 2017. An explorative study of the mobile app ecosystem from app developers' perspective. In *Proceedings of the 26th International Conference on World Wide Web*. 163–172.

[116] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets. In *2018 Internet Measurement Conference (IMC '18)*.

[117] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. 2006. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. *SRUTI* 6 (2006), 31–36.

[118] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.

[119] Shengqu Xi, Shao Yang, Xusheng Xiao, Yuan Yao, Yayuan Xiong, Fengyuan Xu, Haoyu Wang, Peng Gao, Zhuotao Liu, Feng Xu, et al. 2019. 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*. 2421–2436.

[120] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Peng Liu. 2014. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 25–36.

[121] Yury Zhauniarovich, Olga Gadyatskaya, Bruno Crispo, Francesco La Spina, and Ermanno Moser. 2014. FSquaDRA: fast detection of repackaged applications. In *IFIP Annual Conference on Data and Applications Security and Privacy*. 130–145.

[122] Min Zheng, Patrick PC Lee, and John CS Lui. 2012. ADAM: an automatic and extensible platform to stress test android anti-virus systems. In *International conference on detection of intrusions and malware, and vulnerability assessment*. 82–101.

[123] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. 2013. Fast, scalable detection of piggybacked mobile applications. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 185–196.

[124] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. 2012. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 317–326.

[125] Zakiah Zulkefli, Manmeet Mahinderjit Singh, Azizul Rahman Mohd Shariff, and Azman Samsudin. 2017. Typosquat Cyber Crime Attack Detection via Smartphone. *Procedia Computer Science* 124 (2017), 664–671.