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, as well as our tool AppCrazy.

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 unhappy 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 Telegram sneaked into the Google Play store 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...
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., wwwfaebku.com), “Character-omission typos” (e.g., www.faceook.com), “Character-permutation typos” (e.g., www.facebook.com), “Character-substitution typos” (e.g., www.facebook.com), and “Character-duplication typos” (e.g., www.facebook.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, 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 of the official app. There are various approaches to generate fake apps, including cloning the source code, reverse engineering the app, and repackaging the app with a different package name. These approaches allow attackers to create apps that are similar to the official apps, making them difficult to detect.

### App Squatting

App squatting is a relatively new and emerging threat in the mobile app ecosystem. It involves attackers registering app names or package names that are similar to existing apps, in an effort to capture some of the traffic going to the original apps. App squatting can be achieved using various techniques, such as domain squatting, typosquatting, character manipulation, and homograph-based squatting.

### AppCrazy

AppCrazy is a tool that we have developed to systematically generate confusingly similar app names and package names. Given an Android app as the input, AppCrazy automates the generation of squatting identifiers for mobile apps. AppCrazy supports various typo-generation models, including “Missing-dot typos”, “Character-omission typos”, “Character-permutation typos”, “Character-substitution typos”, and “Character-duplication typos”.

We have generated 3,283 potential “squatting names” for 10 popular apps, and discovered 10,553 squatting apps. These findings indicate that app squatting is a highly prevalent threat in the mobile app ecosystem.
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):
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 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 “Facebook1”.
3. **Vowel Character Deletion:** Deleting one or more vowel characters, e.g., “Facebook” into “Facebook”.
4. **Vowel Character Substitution:** Replacing a vowel character with one of the other four vowel characters, e.g., “Facebook” into “Facebook1”.
5. **Double Character Insertion:** Inserting the same character between two consecutive identical characters, e.g., “Facebook” into “Facebook1”.
6. **Double Character Deletion:** Deleting one or two characters that are consecutively identical, e.g., “Facebook” into “Facebook1”.
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 “Facebook1”.

**Combotsquatting 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 paypay-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

Figure 3: The distribution of squatting apps across models.

*Update*. App combotsquatting differs from other forms of app squatting in two fundamental ways: (1) combotsquatting 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 combotsquatting 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.

### 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 Rearrangement”, 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 string, 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...
We first investigate the distribution of squatting apps in the wild (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.

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.

Note that three of them are game apps and two of them are social apps, which suggests these are key targets.

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

<table>
<thead>
<tr>
<th># apps (%) targeted by at least</th>
<th>1</th>
<th>10</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>fake apps</td>
<td>343 (80%)</td>
<td>216 (31%)</td>
<td>66 (15%)</td>
</tr>
<tr>
<td>squatting apps</td>
<td>274 (64%)</td>
<td><strong>106 (25%)</strong></td>
<td>20 (5%)</td>
</tr>
<tr>
<td>Total</td>
<td>375 (88%)</td>
<td><strong>236 (55%)</strong></td>
<td>69 (16%)</td>
</tr>
</tbody>
</table>

**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.
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.

5.3 RQ2: App squatting vs. App Popularity

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.

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 apps are distinct.

6 CHARACTERIZING THE IMPACT

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 apps are distinct.

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.
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 9,390 different versions across the markets. 1,500 squatting apps (with 897 distinct package names) have even been published on 29 different markets. We have identified various aggressive cases. For example, "Faceboook" with the squatting package name "com.bryan.facebook" was found in Google Play, MyApp, Wandoujia, AnzhuoApk, OPPO and 25PP. "WhatsApp" with the squatting package name "com.gbwhatsapp" was found in Google Play, MyApp, Wandoujia, AnzhuoApk, OPPO and 25PP.

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 5 apps with the largest number of squatting attacks across the markets in Table 9. Each of these heavily targeted apps has 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.facebook" was found in Google

---

*MD5:6a69fa5aa4d89932d49a99aa3270023

MD5:8c328159199d1615edac56e8fbf6ez2
Table 9: Top 3 targeted apps according to the number of squatting apps across stores.

<table>
<thead>
<tr>
<th>App Name</th>
<th>Package Name</th>
<th># Sq Apps</th>
<th># Stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angry Birds</td>
<td>com.rovio.angrybirds</td>
<td>537</td>
<td>28</td>
</tr>
<tr>
<td>Spider-Man</td>
<td>com.gameoft.androidANMPGloftJBHM</td>
<td>248</td>
<td>29</td>
</tr>
<tr>
<td>Youtube</td>
<td>com.google.android.youtube</td>
<td>232</td>
<td>24</td>
</tr>
</tbody>
</table>

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 correspond to the squatting versions rather than the legitimate version. 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 Telegram) 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 (EP-SRC EP/N510129/1) and grant EP/P025374/1.
REFERENCES


[95–109].


Bohm T. 2014. Letter and symbol misrecognition in highly legible typefaces for

Hu and Wang, et al.