2018 Master's Thesis

# Scavenging the Remains: A Measurement Study of the Abandoned Internet Resources inside Mobile Apps

Supervisor: Prof. Tatsuya Mori
Research Guidance: The Research on Networked Systems

A Thesis Submitted to the Department of Computer Science and Communications Engineering,
the Graduate School of Fundamental Science and Engineering of Waseda University
in Partial Fulfillment of the Requirements for the Degree of Master of Engineering
July 24th, 2018

Student ID: 5116FG21-1

Elkana Getaloid Pariwono

**Abstract**

This study aims to understand the threats caused by abandoned Internet resources used by Android apps. By abandoned, we mean Internet resources that support apps that were published and are still available on the mobile app marketplace, but have not been maintained and hence are at risk for abuse by an outsider. Internet resources include domain names and hard-coded IP addresses, which could be used for nefarious purposes, e.g., stealing sensitive private information, scamming and phishing, click fraud, and injecting malware distribution URL. As a result of the analysis of 1.1 M Android apps published in the official marketplace, we uncovered 3,628 of abandoned Internet resources associated with 7,331 available mobile apps. These resources are subject to hijack by outsiders. Of these apps, 13 apps have been installed more than a million of times, a measure of the breadth of the threat. Based on the findings of empirical experiments, we discuss potential threats caused by abandoned Internet resources and propose countermeasures against these threats.

# Contents

# List of Figures

# List of Tables

# Chapter 1 Introduction

Access to the Internet enables mobile applications (apps) to offer various rich features such as voice recognition, real-time cryptocurrency chart fetching, and personalized weather forecasts notifications. The working principle behind these features is that huge and complex computational task/data can be stored on the server-side rather than inside each mobile device. This transfer of service/data to the network enables the mobile apps to be light-weight while keeping the rich functionalities because the developers can make use such services/data via API. Moreover, the transfer of service/data eases the development of Internet-connected mobile apps. Consequently, it is common for mobile apps to require Internet permission. All these services that support mobile apps run on top of Internet resources (e.g., domain name, public IP address, or cloud server). In RFC-4085 [1] domain name and IP address is also called Service Identifier.

The problem that we addressed in our study is related to Internet resources used by a mobile app that may be abandoned and therefore at risk for attack even though the app is still available on a mobile app marketplace. In the following, we present how Internet resources are abandoned. For instance, to keep Internet resources such as domain name and server in the cloud running, developers need to pay and maintain the resources. However, once an app is published, the developer does not tend to maintain it. This appears to be a common trend in mobile app development. Derr et al. [2] conducted a survey on the developers of Android mobile apps and found that 78% (158 out of 203) of developers do not regularly update their published app. Moreover, 66% of the developers are not full time developers. These statistics imply that the lack of cost and maintenance is likely to happen on the published apps; hence, the Internet resource used by the apps will also suffer from the lack of maintenance. Such resources will be automatically released after several periods of time and the change of resource ownership could happen, i.e., the resources are abandoned.

As we will discuss in Section 7, the problem of generic abandoned Internet resources has been studied by researchers [3, 4, 5]. Although these previous studies have addressed the threats of abandoned Internet resources, no such study has been conducted on mobile apps and their users, who are the potential victims of hijacked resources. In addition, the impact of a change in ownership of an Internet resource depends on by whom and how the resource was used. Thus, the existence of abandoned Internet resources used by mobile apps raises a new security problem to be addressed.

In order to shed light on the problem, we conducted a large-scale empirical study to understand the potential threats of abandoned Internet resources used by mobile apps. By searching through hundreds of thousands of Android apps, we were able to confirm the existence of these resources inside the code. To understand how Internet resources are being used, we took samples of some of the resources, acquired the resources and collected incoming data from users who had apps that made use of the resources. We discovered that even though a resource has been abandoned and the app is no longer updated, there is still a lot of traffic from mobile users. Finally, we analyzed the threats and present generic attacker models that could be used to exploit this vulnerability.

The contributions of this work can be summarized as follows:

- We highlight a new security problem related to abandoned Internet resources from the per-

spective of a mobile platform and its users. We identify all possible types of abandoned Internet resources that might exist inside mobile apps.

- Our large-scale empirical study revealed that the threat caused by abandoned Internet resources used by mobile app is real and pervasive. We found 3,628 abandoned Internet resources, which are used by 7,331 apps in our measurement study. Among them are 13 apps with 1,000,000+ installations. Our study further revealed that abandoned resources used by third party library had the most widespread impact.
- Our measurement study of these resources revealed several abandoned resources that receive a lot of traffic from Android users and even cross platform users such as iPhone users. In addition, we observed personally identifiable information such as user device ID, and geolocation, which could be used to track the owner of the device.
- As previous studies have noted, the root of the problem is the change in ownership of services that are caused by mobile apps. We discussed possible mitigations of this problem for mobile apps stakeholders.

**Roadmap.** Chapter 2 provides background information on the types of abandoned Internet resources. Chapter 3 describes the methodologies that we used for our measurement and analysis. In Chapter 4 and 5, we present our findings through large-scale measurement and several case studies that demonstrate the threats, respectively. Chapter 6 provides a discussion of the limitation of our approach, possible countermeasures against the attack, and the ethical considerations. Chapter 7 summarizes the related work, and Chapter 8 concludes the paper with a summary.

# Chapter 2 Background

## 2.1 Internet-assisted Apps

To provide rich features, mobile apps make use of services that reside on the Internet. Figure 2.1 illustrates how a mobile app is supported by Internet resources. The app is a news app that gets currency exchange information from a public web service (`currency.example.org`). It also fetches advertisement through ad library (`adserver.example.com`). From the perspective of the ownership/types of the service, we can divide the services into two categories as shown in Figure 2.3; i.e., the private service managed by an app developer (first party) and public service managed by third-party providers.

A Private service is usually offered as a backend service to support the main function of the app. App developers create, manage, and own the service/resources. Running such a service on top of a cloud server is a common approach as it is usually easier to have scalability. Also, developers do not need to maintain physical hardware. For the ease of changes, contacting the server using a domain name is common practice. The developer can change the IP address of the server without having to update the source code.

In contrast, a public service is created, managed, and owned by third-party providers. To access the public service, it may require app developers to include their library (e.g., Facebook Android SDK [6] and ad library); the library will then handle communications with the server. Some other type of public services can be used directly with HTTP calls and authentication key. Typically, the service is a REST API service that communicates with the apps by using HTTP or HTTPS for more secure data transportation. The type of service used by mobile apps is not limited to API server.

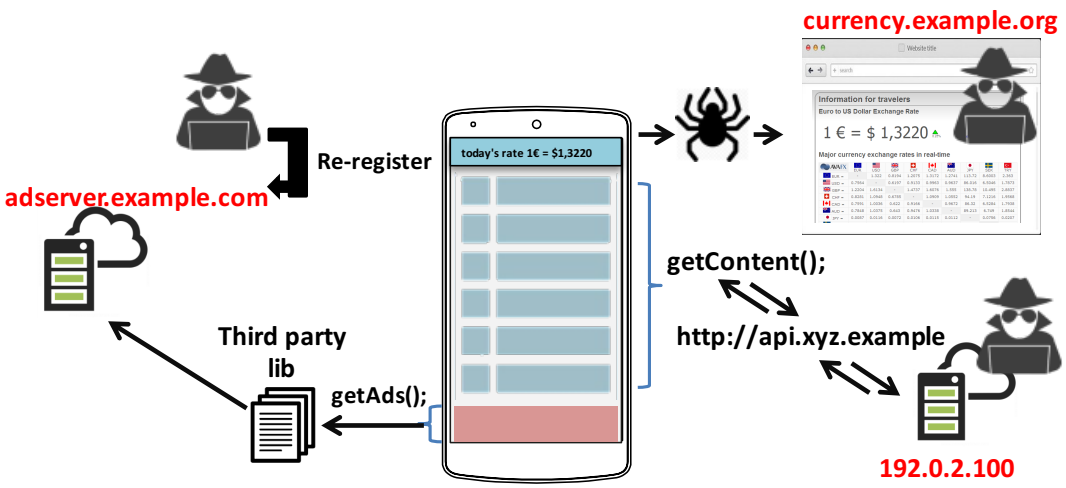


Fig. 2.1 An Example of an Internet-assisted App.

Certain type of apps named Mobile web apps load the content of specific websites into the app by using WebView. Thus the service can also be in the form of a website and that can also be divided within the public service category.

## 2.2 Hijack-able Internet Resource Used by Mobile App

A domain name associated with a mobile app could expire if the owner does not renew it. Once it expires, the change in domain name ownership may occur and a new owner assumes control of all operations performed by the mobile app through the server using that domain name. Figure 2.2 illustrates how an adversary can get her/his hand on the abandoned Internet resource. The developer of the app might realize that some of the app features do not work as intended because the Internet resource that supports their functions is no longer alive.

While developers may decide to fix the problem, Mutchler et al. reported in [7] that a majority of mobile developers do not maintain their apps once published. This means that many of the apps with expired domains will likely remain unfixed. The following are three types of resources found inside Android apps that can be taken over by the attacker:

**Expired Domain**. When a domain expires and the owner of the domain fails to renew it, another person could re-register the domain. As discussed in many previous studies, the purpose of re-registering are sometimes related to malicious activity. In this research, we focus exclusively on expired domains used by mobile apps and the potential threat to mobile users.

**Parked Domain**. An expired domain name might end up as a parked domain, which is an active domain that is available for purchase and may be used in the meantime by the parking domain service for pay-per-click advertisement. If the domain was previously used by mobile apps, there may be a lot of traffic coming to the domain that will raise its price. While these types of domains are expensive, there still exists the possibility of hijacking a mobile app service through a domain purchase.

**Dangling Records (Dare)**. Liu et al. [5] are the first to introduce the term, which refers to domains that are still valid but associated with resources that are no longer alive and can be obtained by other persons. One type of Dare is the IP address in Cloud Dare. The domain name points to an IP address on a cloud (e.g., Amazon EC2 and Microsoft Azure). Once a pre-owned IP address is released, another person can obtain the IP address and abuse the domain. In practice, the cloud server is an ideal option for running a backend service, especially for a startup company. The developer can gradually increase the server resource once the app gains popularity. However, if the business does not do well, the leased cloud server is the first to be abandoned as it costs more and has a shorter lease term than the domain.

**Hardcoded IP Addresses in Cloud**. It is also common for mobile apps to connect to a service using an IP address. We illustrate such an example with Listing 2.1. For the price of losing flexibility with a change in server address, the app will run faster, since there is no need to resolve the domain name into the IP address. As in the case of dangling records, a cloud IP address can be obtained by an attacker once it is released.

Listing 2.1 Example of Hardcoded IP Address in Android Code.

```
public static String _activity_create(boolean p6){
  String v0_1 = new anywheresoftware.b4a.samples.httputils2.httpjob();
  v0_1._download("http://54.214.241.xxx/fugleshow.db");
}
```

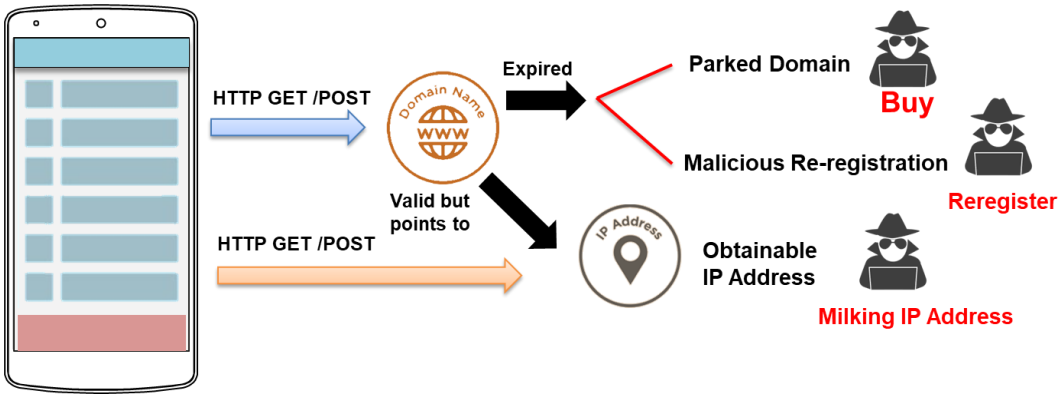


Fig. 2.2 Hijacking Internet Resource Used by Mobile Apps.

## 2.3 Widespread Impact

As shown in Figure 2.3, there are three types of owners of Internet resources that support mobile apps. Depending on the resource owner, the effect of a hijack could vary:

**Private Service**. The same developer that developed the mobile app manages the resource. Hijacking the resource will affect those apps developed by the developer which is connected to this resource. The worst case scenario would be if the developer developed a lot of apps that are connected to this private service. A developer could also have a domain with subdomains as the backend service for each type of app created. The attacker only needs to obtain this domain to take control of all the apps including those connected to the subdomains.

**Third-Party Library**. A third-party library uses resources in the Internet to support its functionality. The library might be used by a lot of different apps. This means that if an attacker obtains the Internet resource, then all apps that use the library can be attacked. An example of this kind of resource is the advertising library. Usually, to insert an advertisement in the app, the developer includes the advertisement library, which manages the process for contacting an Internet server to retrieve the ad. If an attacker gets hold of the server by re-registering the domain or obtaining the server IP address, then the attacker can take control of all apps that use this library.

**Third-Party Service**. Mobile apps could also consume a publicly open service that is managed by another developer. This could be a service created solely to support specific features of mobile apps (e.g., push notification server and mobile analytics service). It could also be in the form of a general service such as a generic website that contains relevant information or feature (URL shortening website, RSS Feeds of news website, etc.). The owner might not know the service is used by a mobile app. This kind of service is also subject to abandonment [5]. When the owner of the service abandons the service, the mobile app developer who is still interested in maintaining the app has to switch to another service with the same features or information. A third-party service differs from a third-party library in the sense that developers do not have to include any library in their apps.

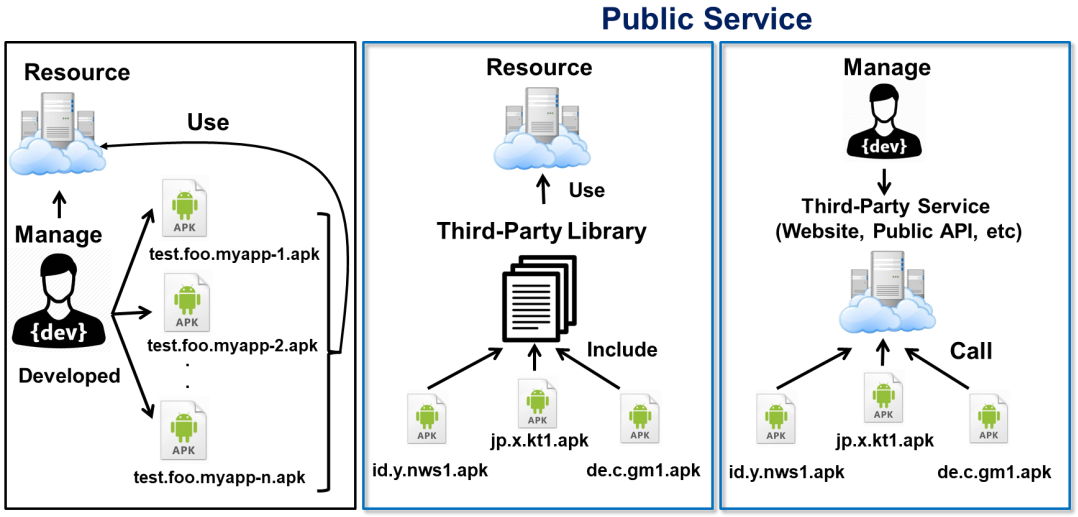


Fig. 2.3 Possible Owners/Types of the Internet Resources. Left: Resource Owner is an App Developer (Private Service), Middle: Resource Owner is a Developer of the Third-Party Library (Public Service), Right: Resource is Owned by a Third-Party Provider (Public Service).

# Chapter 3 Methodology

So far we have discussed the kind of abandoned Internet resources that might be used by mobile apps, our next step is confirming the existence of abandoned Internet resources associated with mobile apps by conducting a measurement study. An overview of our methodology is provided in Figure 3.1. From this study, we attempted to understand the security implications of abandoned Internet resources.

## 3.1 Dataset

Because the main cause of abandoned Internet resources is lack of maintenance, it is likely that we will find abandoned Internet resources by analyzing apps that have not been updated for a while. To this end, we used the Playdrone dataset published by Viennot et al. [8]. They collected over 1,100,000 of free Android apps from Google Play in 2014 and later published the apps for use in the research community. Since the dataset is a snapshot of Google Play in 2014, the most recent apps in the dataset are at least three years old when we conducted our study.

Using this dataset, we first selected all the apps that had Internet permission. As our interest was to demonstrate the real threat of abandoned Internet resource used by mobile app, we only included apps that were still available on Google Play in our analysis. We searched the package name of application in Google play to see if it were available, reviewed the web page, and eliminated the apps that were no longer available in Google Play. We also confirmed that the app in our dataset was the same as the one currently published in Google Play by comparing the published date and version code. We did this check to ensure that the apps have not been updated since 2014. The published date was available in the web page, while the version code was obtained utilizing API calls to Google Play, leveraging the Python library provided in [9]. We also collected the name of the app developer.

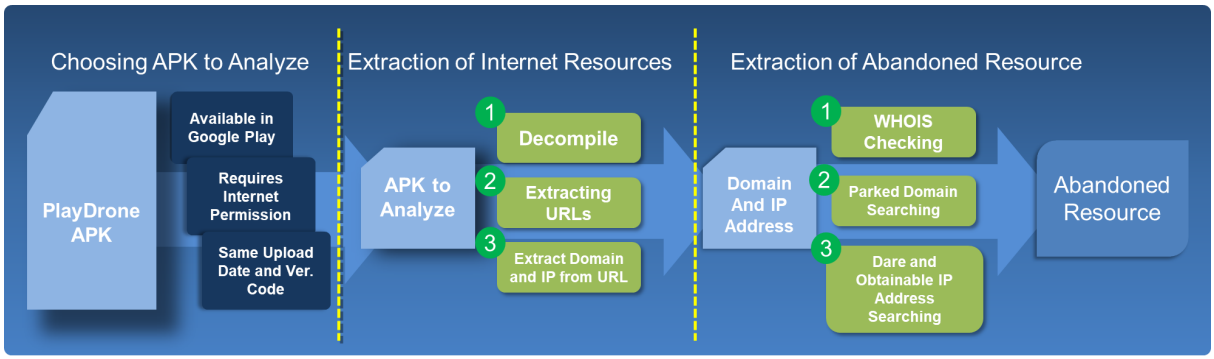


Fig. 3.1 Methodology Overview.

Of the 811,679 APKs in our dataset which were still available, in Google Play, 585,089 APKs require Internet permission. The version code analysis resulted to 180,865 of the APKs that did not give information about version code, meanwhile 292,788 APKs had different version code with APKs in our dataset. Technically, when a developer updates an app, they have to update the version code, upload the APK to Google Play, and the last update date will be updated automatically. However, we found 7,595 apps that had same version code with our dataset but had a more recent last update date. We excluded these APKs. In the end, we ended up with 103,841 APKs that had same version code and same update date.

## 3.2 Extracting Abandoned Internet Resource

Our next task was to check whether the 103,841 APKs contained abandoned Internet resources that could be hijacked. To this end, we developed a tool using the Androguard framework [10] to decompile and extract from the APKs all hardcoded URL. We note that the static analysis suffered from certain limitations. We will discuss this issue in Sections 4.5 and 6.2.

In many the cases, we could infer the kind of services that run on top of the resource by looking at the URL. For example, a URL like `http://foo.test/rssFeed` is an RSS source. We also extracted the line of code, function name, class name, and package name of the class that contained the URL. We excluded schema URI such as `http://schemas.android.com/apk/res/android`. We were also aware that the developer in certain instances obfuscated the code before publishing the app. We filtered those out as obfuscated code is usually indication of a malicious app, whereas our focus was on legitimate apps that had abandoned Internet resource.

We continued with extracting the hostname from URLs. If the hostname part was an FQDN, then we extracted its effective 2LD (e2LD) by using the Public Suffix List [11]. In cases where the hostname part was an IP address we left as is. From the distinct domain, we searched for abandoned Internet resources using the following methods:

**Expired Domain**. By sending a WHOIS query, we checked whether a domain expired or was deleted. Then, we checked whether the domain was available for purchase through GoDaddy Domain Availability API [12].

**Parked Domain**. To determine whether a domain was a parked domain, we used the same technique as in [13]. Technically, when the owner of a domain wants to convert the domain into a parked domain, the domain is configured in a manner defined by the domain parking service (e.g., the domain's Name Server (NS) is made to point to the domain parking service's NS). Vissers et al. studied 15 popular domain parking services and provided known NS record configuration for each service [13]. We followed the provided information to search for parked domain names.

**Dangling Records (Dare)**. We used the same methodology as Liu et al. [5] to search for IP addresses in Cloud Dare. We first determined whether a domain pointed to an IP address in EC2 [14] or Azure [15] and scanned the IP address. When performing a port scanning to the IP address of the domain, we prioritized the custom port numbers found in the URL. Otherwise, we assumed that the app contacts the server by using the standard HTTP Port 80 or HTTPS Port 443. We scanned those ports to confirm whether the host was alive. The only difference between our method and Liu's was performing a second scanning two weeks after the first scanning. In our research, we only focused on dangling records associated with EC2 or Azure.

As for any hardcoded IP address found in the URL, we check whether it fell within the EC2 or Azure IP address range. If so, we performed the same scanning methodology used to search dangling records.

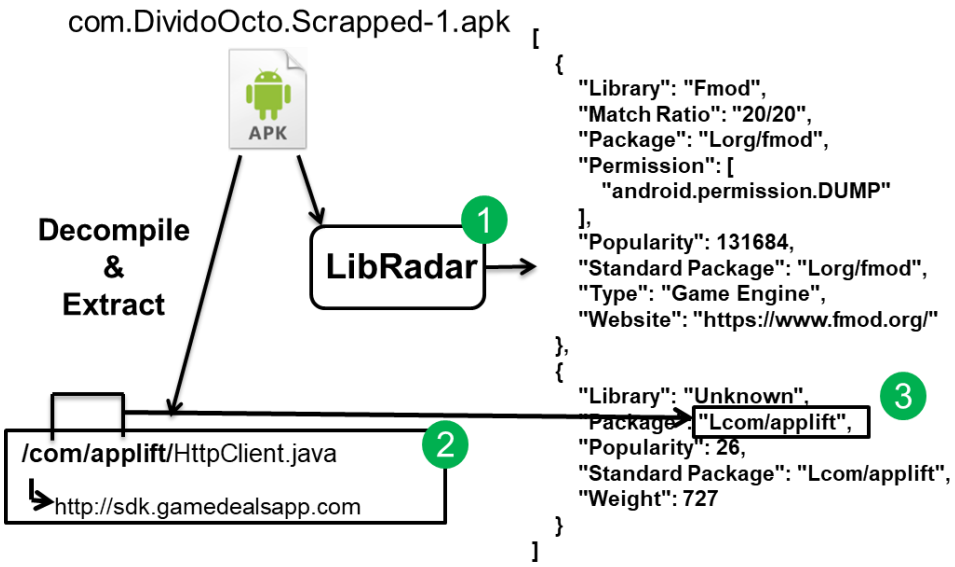


Fig. 3.2 Categorizing Resource into Third-Party Library Domain.

## 3.3 Classifying The Resources

In order to understand the widespread potential of hijacking resources, we classified resources into one of the three categories: private service, third-party library, and third-party service. Figure 3.2 illustrates how we determined whether a resource was used by a third-party library. By leveraging LibRadar [16], we were able to extract information on the third-party library used by a particular APK. LibRadar provided information on the package name of the third-party even when obfuscated. We compared the package name of the resource found by our tool with the package name in LibRadar; if they matched, then the resource was deemed to be used by the third-party library.

For classifying domains into private service or third-party service, we relied on several heuristics as illustrated in Figure 3.3.

**Step 1**: If a resource was used by several APKs, then we checked whether the APKs had the same base package name and was published by the same developer by looking at the developer information from the crawling Google Play page.
**Step 2**: We continued checking whether the resource was found inside the base main package.
**Step 3**: If it matched and the resource was a domain, then we checked whether the domain resembled the package name or developer name by using Fuzzy String Matching provided by [17]. This library employs Levenshtein Distance to calculate the distance between two strings.

We are also aware of the case where the domain did not resemble the package name. In these instances, we checked not only the domain but also the URL. For the most part, we were able to find parts of the URL that resembled the domain and package name/developer name. In one case, we found several APKs (e.g., `appinventor.foo.x`, `appinventor.abc.yzx`, `appinventor.lll.mnl` ) that used the same domain (e.g., `foo.test`), and were developed by the same developer (MMDeveloper); however, only one APK had the base package name

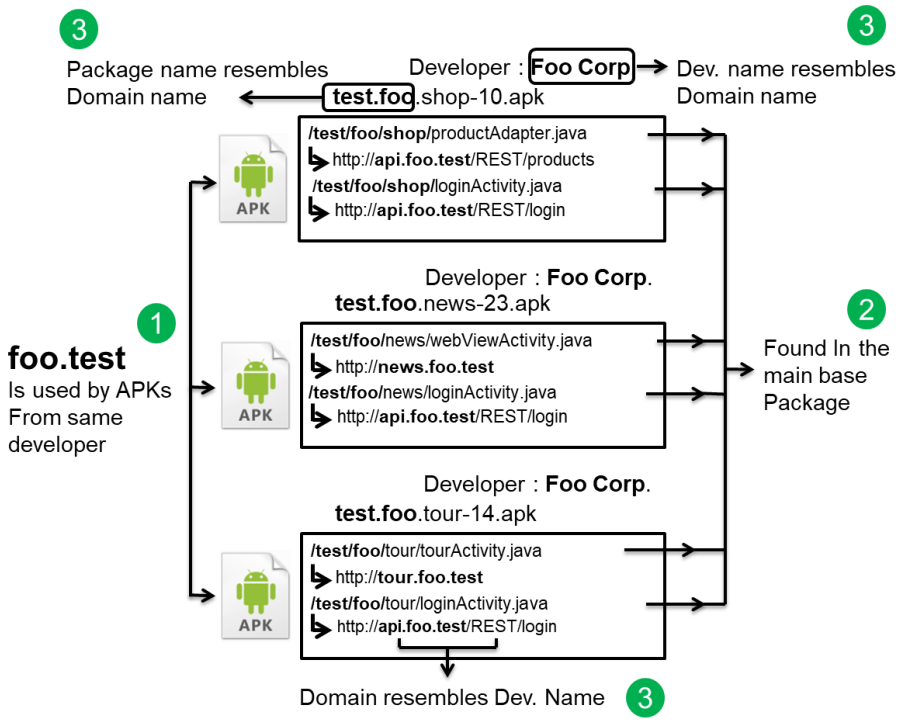


Fig. 3.3 Categorizing Resource into Private Service Domain.

that resembled the domain name (`appinventor.foo.x` and `foo.test`). In this case, we categorized it as a private service. It appears that at the beginning the developer bought this domain to support the functionality of this one particular APK, however after that they use this domain to support the other APKs. Once all these criteria are fulfilled we can certain that this domain was previously managed by the developer of the APKs.

For the remaining domains, which did not meet the prescribed criteria, we suspected that they were third-party services. Again, we relied on several heuristics as provided in Figure 3.4:

**Step 1**: If the domain was used by more than 1 APK, but the APKs did not have same base package name, we searched the APKs to determine whether they used it in the main package. Knowing that the domain was not a private service domain (because the domain name did not resemble a package name or developer name), then we assumed that the domain was likely a third-party service.

**Step 2**: We also checked the URL to help determine what kind of service ran on top this domain (`http://www.foo.test/rssFeed`, `http://example.com/service.asmx?WSDL`, `http://example.org/wp-content`, etc).

**Step 3**: If the subdomain of a domain in use is www, then most likely it was a website for which certain information or certain features were used by the apps crawl. We consulted `archive.org` to confirm this.

For domains that could not be classified using this method, we had to classify them manually by conducting a Google search or doing a manual code inspection. This method also failed with IP address and Dare domains in EC2 with `ec2-ip.compute-1.amazonaws.com` format. We did a manual inspection of the source code, URL structure, and any other breadcrumbs.

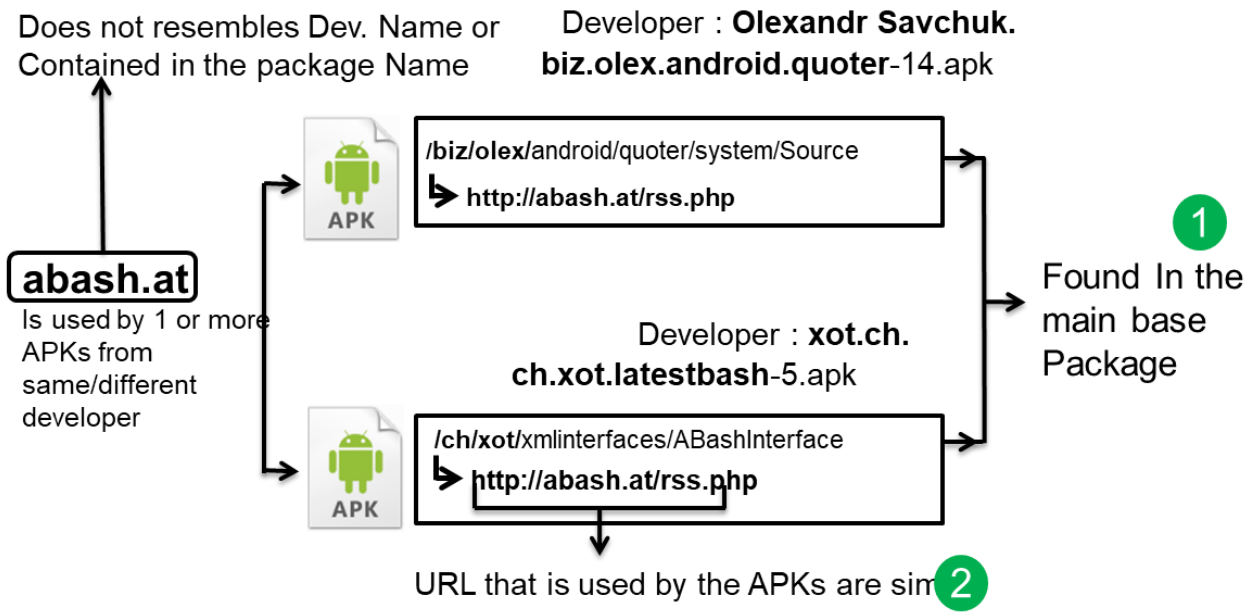


Fig. 3.4 Categorizing Resource into Third-Party Service Domain.

## 3.4 Obtaining the Abandoned Resource

After we confirmed the existence of an abandoned Internet resource used by mobile app, we attempted to find out whether the app still had users by obtaining the Internet resource. Before obtaining the domain, we first checked the last query timestamp of the domain by using Passive DNS. For expired domains we re-registered the domains. For parked domains, we purchased the domains from the domain parking service that owned them. For Dare, we milked the IP address from EC2 by launching a tool similar to IPScouters [5]. All the limitation set in [5] were also applied in our experiment. This IP address milking activity lasted two weeks. While in [5] all acquired IP addresses were released, even when they were associated with dares, we kept the IP address and later launched the instance on that IP address. We set up a web server on the resource and captured all traffic that came to the server. We relied on user-agent string found in the HTTP Header to distinguish whether traffic came from a mobile user. We sent the user-agent string to service in [18]. We also used this service to filter traffics from crawlers.

# Chapter 4 Measurement Results

From the measurement study that we conducted from June to November 2017, we found that abandoned Internet resources used by Android apps existed and took on several forms, which we discussed earlier. In this section, we provide a high-level overview of our results. We then describe the characteristic of the resource with respect to the three major models. We also present the impact of the found abandoned Internet resources as well as findings on bogus domains. Finally, we discuss the threats to validity.

## 4.1 Abandoned Internet Resource in the Wild

Table 4.1 presents the number of abandoned Internet resources, which were found inside our dataset of Android apps. In total, we found 3,628 hijack-able abandoned Internet resources contained in 7,104 APKs. While this number is smaller than the total number of APKs reported in the table, there are many APKs that use more than one type of resource in their code.

Based on the numbers, it is evident that the expired domain types are the most prevalent abandoned Internet resource found in mobile apps. Under the expired domain number, we included 350 domains (connected to 662 APKs) that have past the expiration date and is now in the Redemption Grace Period (RGP), after which it is available to the public for purchase. With respect to parked domains, the number of APKs using these domain are 3.5 times higher than the number of domains. It explains why these are parked domains as they are high traffic domains used by mobile apps.

The number of Dares in EC2 and Azure with its corresponding APKs are almost equal because most of Dares are used by one APK. Given that a host in EC2 is automatically assigned a domain name with `ec2-ip.compute-1.amazonaws.com` format, we found that there are a lot of domains in this form which was found among Dare in EC2. Regarding Dare in Azure, number of APKs is greater (55 APKs) than number of Dare found (56 Dares). This is due to the existence of an APK which uses 2 Dare domains in its code, the domains are `analytics.tapcontext.com` and `register.tapcontext.com`.

We also found several hardcoded IP addresses in EC2. Based on scan results, we concluded that

Table 4.1 Abandoned Internet Resource in the Wild. In total, 468 resources are used by more than one APK.

| Type of Resource | # of Resources | # of APKs |
|---|---|---|
| **Expired Domains** | 2,838 | 4,226 |
| **Parked Domains** | 458 | 1,533 |
| **Dares in EC2** | 199 | 187 |
| **Dares in Azure** | 56 | 55 |
| **Obtainable IP Addresses in EC2** | 77 | 1,330 |
| **Total** | **3,628** | **7,331** |

Table 4.2 Abandoned Internet Resource Category.

| Type of Resource | Private Service | 3rd Party Lib | 3rd Party Service | Unknown |
|---|---|---|---|---|
| **Expired Domains** | 1,436 | 51 | 1,271 | 80 |
| **Parked Domains** | 158 | 15 | 265 | 20 |
| **Dares in EC2** | 113 | 25 | 56 | 5 |
| **Dares in Azure** | 19 | 2 | 14 | 21 |
| **Obtainable IP Address in EC2** | 41 | 7 | 29 | 0 |
| **Total** | **1,767** | **100** | **1,635** | **126** |

the IP addresses were vacant, hence obtainable. There was a huge gap between the number of APKs using hardcoded IP addresses and the number of IP addresses itself. This is due to the presence of 1 IP address which is found inside push notification library and is used by more than 1,000 APKs, detailed analysis on this will be discussed on the next section. While there were found hardcoded IP address in EC2 cloud, we did not find any hardcoded IP address in Azure range.

## 4.2 Characteristic of the Abandoned Internet Resources

**Categories**

By using the heuristics introduced in Section 3.3, we classified the abandoned Internet resources into the three categories presented in Section 2.3. Table 4.2 shows the results. We found that the majority of the expired domains were attributed to the private service or third-party service.

In the private service category, we found 381 apps that were developed using an app building tool called AppInventor [19]. These apps load websites by using WebView. Most of the websites are owned by the same developer since the domain resembles to a part of the package name. For example: `appinventor.ai_emailrobind.PriceComparisonUSA-3.apk` and `pricecomparisonusa.com`. Another example is: `ch.xot.latestbash` consume `http://abash.at/rss.php` for one of the source of its content. Many of APKs that use third-party service are apps that load content from RSS feeds, load a web page from a website as a WebView, or scrape other websites. We note that resources that could not be classified with the heuristics were categorized as "unknown." All APKs in this category were obfuscated and neither the domain name nor URL could be used to identify the developer. A search on `archive.org` produced no results.

The number of distinct APKs for each category is presented in Table 4.3. There were 227 apps that belong to one or more categories. Although there were only 100 resources found in the third-party library category, they were used by 2,440 APKs. Hence, a change of ownership for this category of resources would have widespread impact.

We present the top 10 of the most used abandoned Internet resources on Table 4.4. The IP address used by a third-party library that was developed by UrbanAirship is in first place. The library provides push notification service, using the IP address and custom port number of 8090 to access client's configuration. Scanning result of the IP address indicates that custom port 8090 as well as port 80 and 443 are not opened. Thus, we suspect this IP address is obtainable, however even though we tried to obtain this IP address by continuously milking IP address in EC2 for several days, we were not able to obtain this IP address. There is a possibility that the company still keeps this IP address even though they are not using it anymore.

Table 4.3 Distinct APK per Category.

| Category of Resource | # of Resources | # of Distinct APK |
|---|---|---|
| **Private Service** | 1,767 | 2,542 |
| **3rd Party Library** | 100 | 2,440 |
| **3rd Party Service** | 1,635 | 2,187 |
| **Unknown** | 126 | 201 |
| **Total** | **3,628** | **7,331** |

Table 4.4 Top 10 of the Most Used Abandoned Internet Resources.

| Type of Resource | Domain or IP | # of APK | Category | Detail |
|---|---|---|---|---|
| **Obtainable IP address in EC2** | 75.101.249.xxx | 1,115 | 3rd Party Lib | UrbanAirship Push Notification Lib |
| **Parked Domains** | giveapp.jp | 597 | 3rd Party Lib | App promotion Lib |
| **Expired Domains** | socialauth.in | 203 | 3rd Party Lib | SocialAuth [20] |
| **Expired Domains** | c2town.com | 168 | 3rd Party Lib | com.trid.tridad |
| **Expired Domains** | twipl.net | 134 | 3rd Party Lib | twitter4j |
| **Expired Domains** | kryptomens.com | 126 | Private Service | Back End Server |
| **Parked Domains** | testcocoa.com | 123 | 3rd Party Service | domain from cocoam.co.kr app building service |
| **Obtainable IP address in EC2** | 54.211.58.xxx | 92 | 3rd Party Service | Video Ad Service |
| **Expired Domains** | p41techdev.net | 51 | Private Service | Back End Server |
| **Expired Domains** | urbanislandz.com | 49 | 3rd Party Service | Website |

We consider that private service type resources are a more serious threat. In many cases, the app developer (who also manages and owns the resource) makes use of the resource so as to provide the app with a critical feature, such as a web service that runs on top of the resource. An example is a domain named `kryptomens.com` owned by the developer MYAPPHONE SAS. The backend service that runs on top this domain is providing support for the app to run a deprecated Google service called C2DM (Cloud to Device Messaging) [21]. To use this service, the app had to register the device to C2DM service and send the registration ID to the developer server (See Listing 4.1). Among the dangling records that we found, there were a number of these types of web service. We believe that the developer would not have leased a cloud server if there were no intention to run a critical or heavy service on it. We discuss this case study further in Section 5.1.

Listing 4.1 Private Service at *kryptomens.com* (Owned by MYAPPHONE SAS) is Used as Backend Service.

```
public static boolean registerDevice(String p8, String p9){
  try {
    org.apache.http.client.methods.HttpPost v2_1 = new org.apache.http.client.methods.HttpPost
        ↪ ("http://work.kryptomens.com/mofirst/c2dm/adddevice.php");
    java.util.ArrayList v1_1 = new java.util.ArrayList();
    v1_1.add(new org.apache.http.message.BasicNameValuePair("deviceId", p8));
    v1_1.add(new org.apache.http.message.BasicNameValuePair("token", p9));
    v2_1.setEntity(new org.apache.http.client.entity.UrlEncodedFormEntity(v1_1));
    org.apache.http.HttpResponse v3 = com.myapphone.android.modules.push.NetworkCommunication.
        ↪ httpClient.execute(v2_1);
  }
```

### Life Span of the Abandoned APKs

Figure 4.1 shows how long the APKs that contains abandoned Internet resource inside their code have been available in Google Play. This CDF count is based on the last known update of the APK.

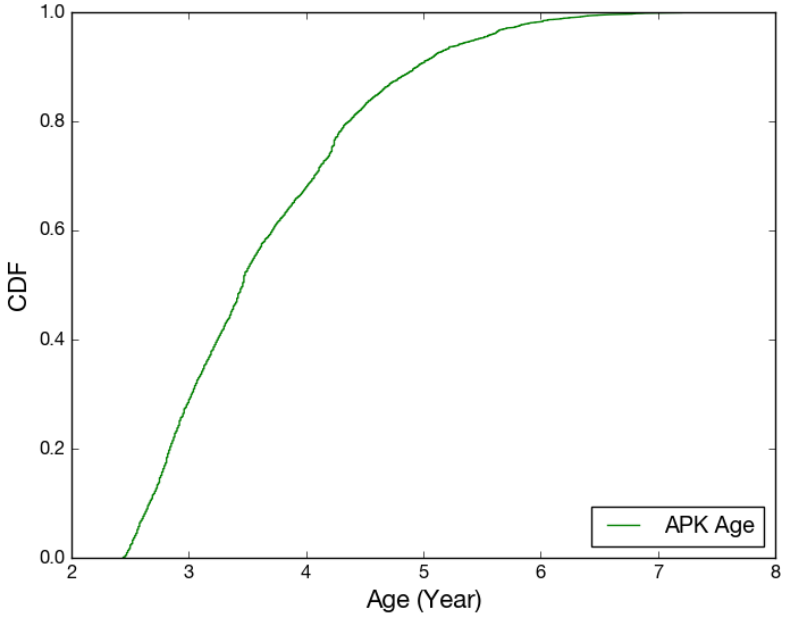


Fig. 4.1 APK Age.

Table 4.5 Requested Permissions of Abandoned APKs.

| Permission | # of APK |
|---|---|
| **WRITE_EXTERNAL_STORAGE** | 4,139 |
| **READ_PHONE_STATE** | 2,837 |
| **ACCESS_FINE_LOCATION** | 2,815 |
| **ACCESS_COARSE_LOCATION** | 2,142 |
| **GET_ACCOUNTS** | 1,905 |
| **CALL_PHONE** | 1,068 |
| **CAMERA** | 1,035 |
| **READ_EXTERNAL_STORAGE** | 587 |
| **READ_CONTACTS** | 507 |
| **SEND_SMS** | 483 |

On average, 50% of the APKs have been not updated for more than 3 years and less than 4 years. It is apparent that the APKs that have abandoned Internet resource have been in Google Play for quite sometime.

**Permissions of Abandoned APKs**

We noticed that there are a number of abandoned APKs requesting dangerous permissions. Based on the dangerous permission which is defined in [22] by Google, we list the top 10 dangerous permissions based on the number of APK that uses it in Table 4.5. As stated on [22], Google defined these permissions dangerou because they grant access to information or data that contains user's private information. As the app might send data containing user's private information to the resource on the Internet, then it is possible for the attacker to obtain this data by hijacking the resource.

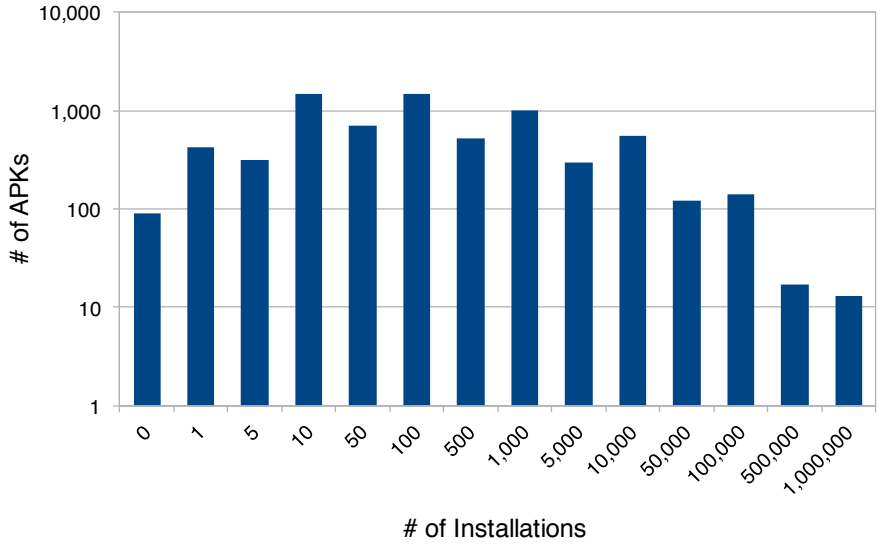


Fig. 4.2 Number of Installations of the Apps that Contain Abandoned Resources.

Table 4.6 Parked Domain Price.

| Domain | Price | # APK | Category |
|---|---|---|---|
| **giveapp.jp** | USD 19K | 597 | 3rd Party Lib |
| **testcocoa.com** | USD 477 | 123 | 3rd Party Service |
| **rank-park.com** | USD 3.7K | 49 | 3rd Party Lib |
| **tovingo.com** | USD 4.6K | 39 | Private Service |
| **nimbosolutions.com** | N/A | 34 | Private Service |

## 4.3 Impact of Hijacking the Resource

To study the impact of hijacking resources, we first studied the popularity of apps that use the abandoned Internet resources. We then looked at the price of the abandoned resources.

Figure 4.2 shows the distribution of the number of installations per app that had abandoned Internet resources. By looking at number of installations, we concluded that most of the APKs were not popular and could explain why the resources were not updated for a long time. There are, however, several apps with more than 1,000,000 installations. There is the possibility that these APKs still have active users and hijacking the abandoned Internet resource would impact them. In Chapter 5, we discuss whether an abandoned Internet resource actually attracts traffic from real users.

For a certain type of resource such as parked domain, one way to measure the number of potential victims (active users of the app) was by looking at the price of the resource, which is an indication of high-traffic usage. We observe that the price of the parked domain which is used by a lot of APKs is exceptionally high. In Table 4.6, we present the price of the top most used parked domain we found. We could not get the price of `nimbosolutions.com` domain, but according to WHOIS result this domain is held by domain parking service named InternetTraffic. In summary, by looking at the price, one can conjecture that all these domains still have a lot of traffic because traffic can be monetized [13].

Table 4.7 Placeholder Domains.

| Domain | #APK |
|---|---|
| **dummy.com** | 1,152 |
| **placeyourdomainhere.com** | 327 |
| **dummyurl.com** | 243 |
| **openuri.org** | 113 |
| **baseurl.com** | 37 |
| **myorg.org** | 34 |
| **mycompanyurl.com** | 32 |
| **example.org** | 15 |
| **yoursite.com** | 11 |
| **someurl.com** | 5 |
| **samplewebsite.com** | 1 |
| **somesite.com** | 1 |
| **some.site** | 1 |
| **yourdomain.com** | 1 |

Under parked domains, we found a domain named `giveapp.jp`, which was formerly an app searching service for the official application in Google Play and Apple Store. They provide a third-party library `jp.co.cayto.appc.sdk.android` and `net.app_c.cloud.sdk`. These libraries have functionalities of providing app ranking and app promotion. These functionalities were associated with the two subdomains of `giveapp.jp`. The first library `jp.co.cayto.appc.sdk.android` used `api.giveapp.jp` while the second library, `net.app_c.cloud.sdk` used `android.giveapp.jp`. This information was displayed as advertisement on the app. As of December 2017, the parked domain is held by Sedo domain parking service with a value of USD 18,870. Domains related to advertising are also commonly found in this category such as `adcube.net` held by Sedo domain parking service and worth USD 4,888. These prices are an indication of high-traffic domains that can pose a certain risk to visitors [13].

## 4.4 Bogus Domains

As a byproduct of our analysis, we found several bogus domains, which were used as placeholders inside the code (Table 4.7). This kind of domain particularly appeared in third-party libraries. For instance, we found `placeyourdomainhere.com` inside OSMDroid (Open Street Map Droid) third-party library.

Looking from the names, the developers are supposed to change the example domain to their own domain. However, they did not change it. The chances are either the developer was unaware of it or they did not use features associated with the bogus domain. We found that most of these domains are parked domains (10 domains). Three of them are available for purchase. We note `example.org` is reserved by IANA [23]. Like the abandoned domains, this kind of bogus domains can also be a source of attacks.

## 4.5 Threats to validity

Although abandoned Internet resources of third-party libraries would have the most widespread impact if hijacked, we were also aware that network operation code in this resource may not be

executed on run time. This is one of the limitation of our work that we discuss in Section 6.2. To validate our approach, we compared the URLs of nine APKs detected by our static analysis with the URLs detected by system that implements the dynamic analysis. For dynamic analysis, we made use of a commercial sandbox that could perform symbolic execution. While there was a difference in the number of URLs found, the number of distinct domains and subdomains found in our tool and sandbox were the same.

There is also the possibility that a resource may not be relevant to the issues raised in this study. For instance, `socialauth.in`, which is ranked 3rd (Table 4.4) was used by the developer for testing connection. By judging from GitHub repository of this library, we are certain that the developer of this library is still maintaining it, since the last commit was quite recent. Even so, if the developer of the apps does not update the library, then the mobile app will be still using the old version library that contains abandoned Internet resource.

# Chapter 5 Threat Case Studies

After we confirmed the existence of the abandoned Internet resource used by mobile apps, next question to be answered is whether there is still traffic from mobile user to this.

## 5.1 Privacy Leak

As we have mentioned earlier, mobile apps may send sensitive information such as user location and device ID to its back-end server. For instance, by milking IP address on EC2, we obtained several IP addresses, one of which is used by a domain named `prod1.magtogo.com`. This domain is owned by developer called MagToGo. According to Google Play, they have developed 76 apps but our dataset only contained one in it. Through Google Cloud Messaging (GCM) Service, this server sent push notification message to its client. For that purpose the app needed to register the device by sending device specific information such as IMEI and Device ID to the domain that runs a REST Service on top of it. We launched an EC2 instance on top of this domain, captured the traffic of this domain for a one-month period and received 5,098 HTTP POST traffic. Based on the user-agent string we counted the number of mobile users and only found 26 distinct Android users and 57 distinct iPhone users. However, inside the HTTP POST body, we found 1,186 unique Device ID. Since we counted distinct users based on the IP address and employed heuristics to confirm that the request was from the same user if it came from the same IP address within one day, then we might have missed a number of users behind NAT IP address. In addition, we found 1,794 APKs requesting for `C2D_message permission` (permission for cloud to device message); in the worst case, these 1,794 APKs might leak some sensitive information such as Device ID and IMEI.

## 5.2 Scamming and Phishing

We assumed that because parked domains have high traffic from mobile users, this type of domain would be a good target for scamming and/or phishing. To test that, we purchased a parked domain named `iphone-english.com` which was worth USD 300. The domain is used by a developer called Fourhalf as a full WebView Activity inside their apps. As far as we could tell, the developer built ten apps using AppInventor [19] service. These apps are language learning education apps that load `iphone-english.com` web page as a WebView. We launched server on it and captured the traffic. We observed that traffic came from both iPhone and Android users. Using the same counting method as we did earlier, we found that even though the app was old (last update in 2012) there were still 93 users from Android and 405 users from iPhone . Since the actual app requires users to login (shown in Figure 5.1 (left)), we conducted a simple experiment to see how many users attempted to login to the app. We set up a simple web login page that had social networks login buttons (Figure 5.1, right) and counted the number of users who clicked the button. As we intend not to collect sensitive information from users, we set the buttons so as nothing happens

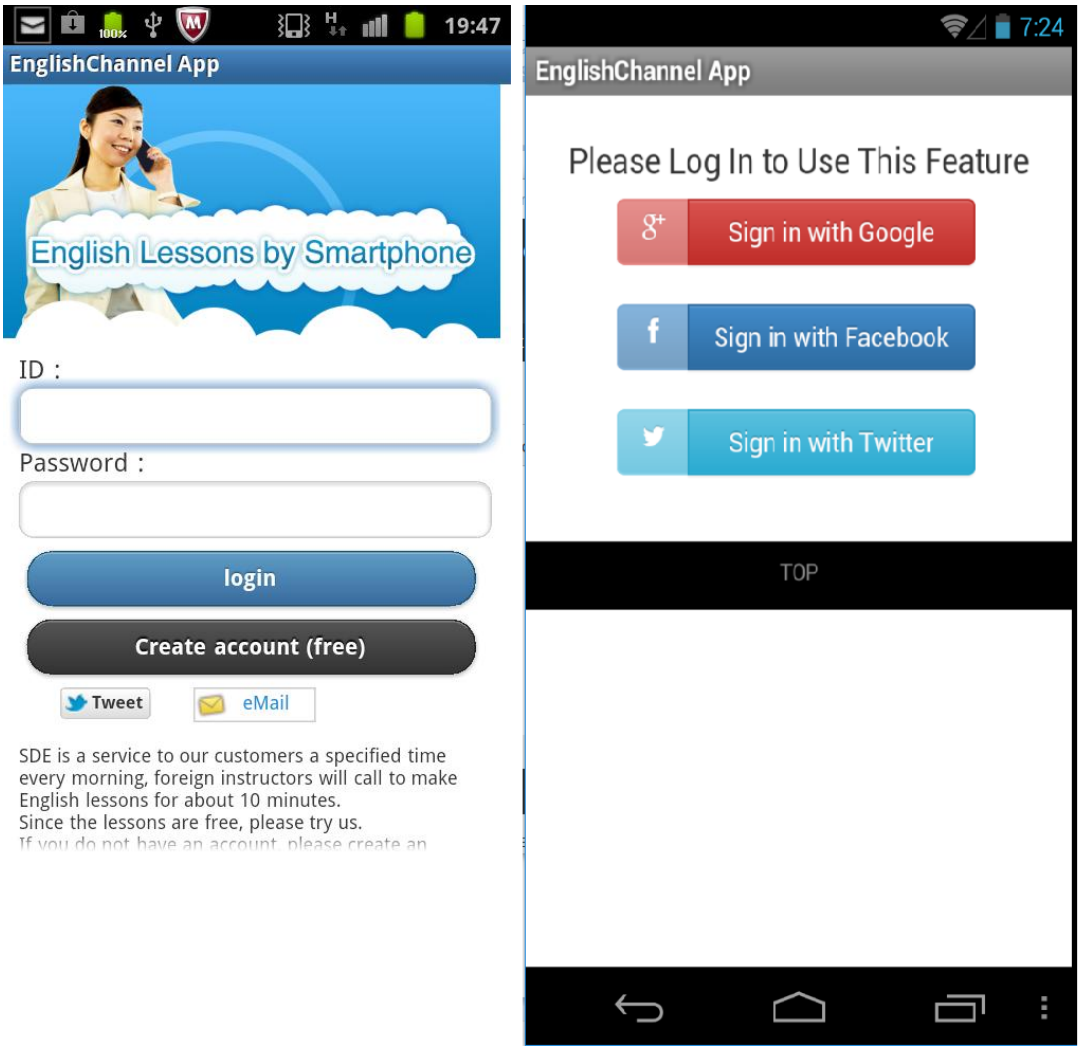


Fig. 5.1 Image on the Left is the Original WebView App. Image on the Right is the App Page for Phishing.

when the user clicked a button.

We found that the 30 users who clicked the button were all Android users. We conducted this experiment within one month period. An attacker could set up a phishing or scamming page for mobile apps that use the Internet resource to load WebView.

## 5.3 Monetizing

Parked domains are highly valuable because of the traffic they bring. An attacker can buy the domain immediately after it is released to public and turn it into parked domain. To increase the chance of making money, the attacker can specifically target the domain from third party library. Rather than make a parked domain out of typosquatting domain, which is a common method, this is a more certain way to make money. Other than engaging in malicious activity, an adversary can resort to this method. Moreover, if the domain is used as a WebView then the app user would be exposed to scamming as reported in [13].

## 5.4 Injecting Malware Distribution URL

This kind of threat may emerge from consuming untrusted third-party service. For example, wallpaper apps load image from expired domains or CDNs by directly specifying the full URL in the code. Once a malicious person re-registers the domain, malware can be put into links disguised as picture links. Another case involved an IP address used by a URL shortener startup service called `fbshare.me`. This service provides a share button that shows the number of times the URL of the current page appears on Facebook. This domain was found inside a blogging app in our dataset. Users can manage their blog by using such an app. After we launched instance on this domain, we observe a lot of mobile user access the short URL, which earlier on was provided by this service. The number of **Android users** and **iPhone users** were **6,633** and **22,289** respectively. A malicious person could distribute malicious software through the URLs. We were aware that this was more a general problem of Dare. If an adversary wants to specifically targeting mobile user, then they can search for Dares used by mobile app.

# Chapter 6 Discussion

Throughout this study, we highlighted the risk and the pervasiveness of abandoned Internet resource used by Android Apps. Since the apps were fairly old, we were aware that not all apps still have users. As we revealed in the previous section, however, 13 active apps with abandoned Internet resources have had installed more than a million of times and some abandoned Internet resources are still accessed frequently from end users. We also believe that regardless the number of users, developers and official marketplace operators should take responsibility for addressing the problem and preventing distribution of apps that contained abandoned Internet resource. In this section, we discuss the possible mitigation of this problem from the perspective of the developer and marketplace operator. We conclude with discussing the possible future direction of this work.

## 6.1 Countermeasures

As stated in former studies, the root problem is the change in ownership of the resource. We propose the following mitigation for cloud providers, mobile marketplace operator, and developers:

**Allowing Inquiry of IP Address Ownership**. Cloud providers can provide a service that allows third parties to inquire about the owner of a particular IP address while keeping the privacy of the owner. To protect personal identity of the owner, cloud providers give the information in their hash value. Thus, developer of the apps and marketplace operator can check from time to time whether a particular IP address has changed ownership or not.

**Monitoring the Internet Resources Used by the Apps**. In the time that the app is published, we suggest that the marketplace operator record all Internet resources used by mobile apps. The market-place operator can check for the ownership change of a resource by employing an algorithm like Alembic [24]. Once ownership change is detected, the marketplace operator can confirm with the developer and suspend the app. When it comes to the domain used by third party library, marketplace operator such as Google Play can put it into a vulnerability campaign such as ASI program [25] and inform all developer that use the third-party library in their apps.

**Performing Authentication on Application Level**. Studies in [26, 27] discuss how a service authenticates the app that request permission to use the service. However, from the case studies we have come across, it is necessary for apps to authenticate the service to which it is connecting. Since the adversary controls the domain and can generate a valid HTTPS certificate, it would be better for mobile developers to implement certificate pinning which is encouraged by Google [28]. By doing this the certificate will be embedded in the application. In this way, when an adversary hijack the backend server through domain registration, and then they generate a new HTTPS certificate, the application will not trust it. However, when the developer updated the certificate, they have to update the application as well. In addition to using HTTPS, mobile developers who own private service and third-party library developers should implement ways for the app to authenticate the

backend service in order to check whether it is connected to a legitimate service or not. We leave the details of implementing this in future work.

## 6.2 Limitations

**Static Analysis** As we adopted static analysis for our study, there are several known drawbacks. First, even though we can detect the abandoned Internet resource used by mobile apps, there is no guarantee that the resource will be executed during run time. Second, our methodology is vulnerable against obfuscated apps. Therefore, this study is biased towards unobfuscated apps. Moreover, since full URL (or even the base URL) can be constructed using various string operation method [29], our approach cannot extract URL constructed in such way. We are also realized that there is the probability that not all URL are called in execution time. In future work, when searching for abandoned Internet resources, we plan to adopt more sophisticated approach such as the Tiger system [30] approach for extracting network-related code and then using this information to trigger network activities on the dynamic analysis. We can thereby increase the validity and reliability of our analysis.

**Tracking Hijacked Resources** In this work, we were not able to measure how many resources have been hijacked or how many of them have changed their ownership since the publication of an app. In future work, we intend to employ algorithm such as Alembic [24] or Passive DNS service [31] to detect ownership change made in the past.

**Dataset** Since we use Playdrone dataset, the experiment was biased towards free app. Abandoned Internet resource used by mobile apps should also apply to paid apps. Since the Playdrone dataset is only a snapshot of Google Play free apps on 2014, all the apps are fairly old. There is a possibility that more recent apps are using abandoned Internet resources. Finally, while we focused on Android apps in our study, our approach is applicable to other platforms such as iOS.

## 6.3 Ethical Consideration

We conducted this study according to research ethics principles and best practices [32, 33, 34]. Our scanning servers for dangling records only generated a restricted amount of traffic (i.e., requests to HTTP port 80, port 443, and custom port found in the URL), which did not increase server workload. On resources we acquired, we simply collected requests for resources, did not respond to anything harmful, and did not receive sensitive or personal information. We acquired abandoned Internet resources for the purpose of our research, we followed procedures for responsible disclosure, and we are in the process of reporting them to app/library/service developers.

## 6.4 Abandoned Internet Resources on Another Platform

As mentioned in Chapter 4, we observed that there were traffics from mobile apps on the different platform came to the abandoned Internet resources that we obtained. This is an indication that abandoned Internet resources problem does not only exist on the Android mobile platform but on the other mobile platform as well, such as Apple iOS. We have actually anticipated this possibility before and intended to measure this problem on iOS platform also. However, collecting a large number of applications on this mobile platform is still a challenging problem.

Another of our conjecture is that this problem also exists on IoT (Internet of Things) platform. In order to prove this, we conducted a preliminary study on a small number of firmwares that were used by IoT devices. Table 6.1 presents the information about firmware that were used in this pre-

Table 6.1 List of IoT Firmwares that were Used on Preliminary Study of Abandoned Internet Resources on IoT Platform.

| Vendor | Category | Product Series | Firmware Last Update |
|---|---|---|---|
| **Vendor A** | Wireless Router | Product A-1 | 2015-May-08 |
| **Vendor A** | Wireless Router | Product A-2 | 2015-December-15 |
| **Vendor A** | Wireless Router | Product A-3 | 2016-January-06 |
| **Vendor A** | Wireless Router | Product A-4 | 2016-January-12 |
| **Vendor A** | Wireless Router | Product A-5 | 2016-March-01 |
| **Vendor A** | Wireless Router | Product A-5 | 2016-May-23 |
| **Vendor A** | Wireless Router | Product A-5 | 2016-September-01 |
| **Vendor A** | Wireless Router | Product A-4 | 2017-March-01 |
| **Vendor B** | Wireless Router | Product B-1 | 2016-September-27 |
| **Vendor B** | Network Camera | Product B-2 | 2017-July-24 |
| **Vendor C** | Wireless Router | Product C-1 | 2017-September-13 |

liminary study. We leveraged the same methodology as the one that we used in Android apps measurement study. The result is we found 2 expired domains (`leachsite.com` and `krb4site.com`) and 2 parked domains (`picturemania.com` and `secure-site.com`) inside a firmware that was used by a network camera device from Vendor B (Product Series: Product B-2). Further analysis of the usage of these domains will be our future work. Abandoned internet resources in IoT device poses serious threat because the owner of the device tends not to replace the device as long as it is still working properly. Moreover, if the vendor has stopped their support on the device and does not release firmware update or patch anymore then this problem will remain unfixed in the operational device.

All of the indications and investigation results that are mentioned above shows us that a measurement study related to this problem on another platform is necessary. Thus, our future work is to conduct a large-scale measurement study on the iOS application and also on IoT devices.

# Chapter 7 Related work

## 7.1 Abandoned Internet Resource

Abandoned Internet resource and change of ownership have been discussed in several studies before. Schlamp et al. [3] discussed the problem in terms of abandoned public IP address prefix. One major cause is that companies who go out of business do not clean up their resource properly. In [5], abandoned Internet resource was reported to be pervasive and vulnerable to abuse. It can also be caused by unexpired domains and obtainable resources (Dangling Records).

However, threats caused by abandoned Internet resource associated with mobile app are not discussed in these two studies. The closest to our research is a study conducted by Mutchler et al. [7]. They did a large scale study about mobile web app and found several expired domains used by the Android app. Whereas our research is focused on abandoned Internet resources used by Android app regardless of usage, meanwhile their research looked at the vulnerability in Android WebView system and the possibilities of loading untrusted websites in WebView.

## 7.2 Domain Name Analysis

Previous studies analyzed abandoned and re-registered domain names through the lens of domain registration and monetization processes. Hao et al. focused on the registration process of spammers' domain names to explore the characteristics of domain registrars and domain life cycles [35]. Their measurement showed that spammers frequently re-registered expired domain names and they preferred domain names that had recently expired. Moreover, Hao et al. proposed a domain reputation system called PREDATOR [36] to identify malicious domain names using the features derived from the registration process. The argument is that attackers need to register many domain names to enhance their attack agility, leading to abnormal registration behaviors. Lever et al. introduced the concept of *residual trust*, which relates to the historical reputation of a domain name that is passed down through a change of owners [24]. It was observed that attackers exploited and abused residual trust. Similarly, Lauinger et al. focused on *residual trust* and analyzed large-scale WHOIS data for analysis of re-registrations [4]. They showed that expiration processes differed substantially among domain names. Many re-registrations happened soon after deletion, especially for older domain names. Lauinger et al. also conducted empirical analysis of ownership changes of re-registered domain names [37], demonstrating that a majority of re-registered domain names used domain parking services and hosted only advertisements. The domain names were predominantly used for speculation and monetization purposes and relied on residual traffic. Alrwais et al. reported the systematic measurement study on the dark side of domain parking system based on infiltration analysis, meaning they tracked end-to-end monetization chains on domain names hosted by major domain parking services [34]. The results exposed the practice of click fraud, traffic spam, and traffic stealing during the monetization of parked domain names. Vissers et al. presented an in-depth exploration of the domain parking's ecosystem in terms of the consequences of accessing

parked domain names [13]. They showed that users landing on websites hosted on domain parking services were exposed to various types of malware distribution and scam websites.

Our study brings a new perspective of *abandoned* domain names resulting from ill-maintained mobile apps. We believe that our study enhances understanding of abused domain names that were discussed in previous studies.

## 7.3 Security Analysis of Mobile Apps

Many studies have revealed the security risks of third-party library used in mobile apps. Program analysis of mobile apps requires identifying third-party libraries and separating them from the *host app* in order to obtain accurate analysis results. LibRadar, LibScout, and LibD are state-of-the-art library detection tools that do not rely on whitelists and are publicly available [38, 39, 16]. We employed LibRadar to classify the resources used by apps in Section 3.3 because it provided information on the package name of third-party libraries.

The security risk of third-party libraries mentioned in the past studies were typically classified into unwanted program and vulnerability. Potentially unwanted or harmful apps are known to be attributed to third-party libraries. Andow et al. discovered that more than 1% of the apps in randomly selected apps from the PlayDrone dataset aggressively display advertisements and are attributed to have malicious adware libraries [40]. Chen et al. presented the methodology for locating these libraries. They dissected 140 potentially harmful libraries collected across Android and iOS marketplaces [41]. Backes et al. conducted a longitudinal study of library usage and evolution in apps over time and discovered that known vulnerabilities in popular libraries still remain unfixed in the current top apps in that time [38]. Derr et al. conducted a survey of app developers and performed a large-scale library updatability analysis. They focused on the root causes of *outdated* libraries and proposed actionable remediation for stakeholders [2].

Our study shed light on the *availability* of Internet resources used by mobile apps, which is the new aspect of mobile library analysis different from above studies.

# Chapter 8 Conclusion

This study focused on the threat of abandoned Internet resource used by mobile apps. We conducted a large scale measurement study using 1.1 M of Android apps. We first defined the types of abandoned Internet resource and search for such resources inside Android apps code. We found 3,628 abandoned Internet resources from 7,331 apps that were published in the official marketplace–Google Play. This 3,628 abandoned Internet resources took form in 4 different types–Expired Domain, Parked Domain, Dangling Records, and Hardcoded IP Addresses in Cloud. Moreover, we confirmed that the abandoned Internet resources have different widespread impact once it is hijacked. This widespread impact is influenced by the previous owner of the resources. Furthermore, We categorized the 3,628 abandoned Internet resource that we found into 3 widespread categories–Private Service, Third-Party Library, and Third-Party Service. Our result show that, abandoned Internet resources contained in third-party library have the most widespread impact.

By acquiring and evaluating these resources, we demonstrated that the threats of abandoned Internet resources were: **Real**. Since we confirmed that even when the app has long abandoned, and so does the Internet resources inside, there were traffic from mobile user. **Serious**. An adversary can abuse the resource just by acquiring it, moreover the traffic that comes to resource can contain sensitive information. The victim of this abuses is the user of the application. **Pervasive**. From our measurement study we note that the usage of an abandoned Internet resource can spread across different applications and even across platforms. As remedies of this problem, we proposed a practical solution to be adopted by cloud providers, marketplace operators and developers.

Based on the traffic from the acquired abandoned Internet resources, we observed the potential of this being a cross mobile platform problem. In addition to this, we suspected that this problem exists on IoT platform as well. Thus, we conducted a preliminary investigation on IoT platform and confirmed the existence of this problem on the said platform. Therefore, future work intends to increase the coverage of the measurement, and measure the problem not only on another mobile platform but on IoT platform as well.

Finally, as we can see from the changes in the popularity of apps in a marketplace, one app may gain tremendous popularity, but its popularity may also be rapidly lost. As long as such a structural property exists, the problem of abandoned Internet resources will continue. It is expected that the problem addressed in this thesis will lead to further research towards a fundamental solution.

# Acknowledgement

I would like to express my deepest thanks and appreciation to Prof. Tatsuya Mori for all his supervision and passionate guidance throughout my master study. I would like also to show my gratitude to Dr. Daiki Chiba and Dr. Mitsuaki Akiyama from NTT Secure Platform Laboratories for all their comments, wisdom and assistance during the the course of this research. Thanks are to the anonymous reviewers of ACM ASIACCS 2018 for their thoughtful feedback. My gratitude is also to Dr. Bo Sun for his insightful and valuable advice to this project. I also thank Dr. Mitsuhiro Hatada for letting me use the commercial sandbox tool. I am also grateful to Dr. Dave Plonka for his comments on abandoned Internet resources (RFC 4085). Many thanks are to all members of Networked Security Laboratories that have taught and helped me in a lot of things during my stay in Japan. A part of this work was supported by JSPS Grant-in-Aid for Scientific Research B, Grant Number 16H02832. Lastly, special thanks are to Indonesia Endowment Fund for Education (LPDP) that has provided full scholarship for my master education.

# Achievement

**Domestic Conference**

- Elkana Pariwono, Daiki Chiba, Mitsuaki Akiyama, and Tatsuya Mori. 2018. Measurement Study of the Hijackable Internet Resources inside Mobile Apps. Symposium on Cryptography and Information Security. Niigata, Japan.

**International Conference**

- Elkana Pariwono, Daiki Chiba, Mitsuaki Akiyama, and Tatsuya Mori. 2018. Don't throw me away: Threats Caused by the Abandoned Internet Resources Used by Android Apps. In Proceedings of the 2018 on Asia Conference on Computer and Communication s Security (ASIACCS '18). ACM, New York, NY, USA, 147-158. DOI: https://doi.org/10.1145/3196494.3196554

# Bibliography


[1] Serviceidentifier. https://tools.ietf.org/html/rfc4085section-3.3, 2005.

[2] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pp. 2187–2200, 2017.

[3] Johann Schlamp, Josef Gustafsson, Matthias Wählisch, Thomas C. Schmidt, and Georg Carle. The abandoned side of the internet: Hijacking internet resources when domain names expire. In *Proceedings of the 7th International Workshop on Traffic Monitoring and Analysis (TMA'15)*, pp. 188–201, 2015.

[4] Tobias Lauinger, Kaan Onarlioglu, Abdelberi Chaabane, William Robertson, and Engin Kirda. Whois lost in translation: (mis)understanding domain name expiration and re-registration. In *Proceedings of the 2016 Internet Measurement Conference (IMC'16)*, pp. 247–253, 2016.

[5] Daiping Liu, Shuai Hao, and Haining Wang. All your dns records point to us: Understanding the security threats of dangling dns records. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pp. 1414–1425, 2016.

[6] Facebook android sdk. https://developers.facebook.com/docs/android/getting-started, 2017.

[7] Patrick Mutchler, Adam Doupé, John Mitchell, Christopher Kruegel, and Giovanni Vigna. A Large-Scale Study of Mobile Web App Security. In *Proceedings of the Mobile Security Technologies Workshop (MoST)*, May 2015.

[8] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'14)*, pp. 221–233, 2014.

[9] Google play unofficial api. https://github.com/egirault/googleplay-api, 2012.

[10] Androguard. https://github.com/androguard/androguard, 2016.

[11] Public suffix list python. https://pypi.python.org/pypi/publicsuffix/, 2017.

[12] Godaddy api. https://developer.godaddy.com/doc, 2017.

[13] Thomas Vissers, Wouter Joosen, and Nick Nikiforakis. Parking sensors: Analyzing and detecting parked domains. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS'15)*, 2015.

[14] Aws ec2 ip address range. http://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html, 2017.

[15] Microsoft azure ip address range. https://www.microsoft.com/en-us/download/details.aspx?id=41653, 2017.

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

[17] Fuzzymatching. https://pypi.python.org/pypi/fuzzywuzzy, 2017.

[18] Useragentstring. http://useragentstring.com/pages/api.php, 2011.

[19] Mit app inventor. http://appinventor.mit.edu/explore/, 2017.

[20] Socialauth. https://github.com/3pillarlabs/socialauth, 2017.

[21] Android cloud to device messaging (c2dm). https://developers.google.com/cloud-messaging/c2dm, 2016.

[22] Android permissions. https://developer.android.com/guide/topics/permissions/requesting.html, 2017.

[23] S. Cheshire and M. Krochmal. Special-Use Domain Names. RFC 6761 (Proposed Standard), February 2013.

[24] Chaz Lever, Robert J. Walls, Yacin Nadji, David Dagon, Patrick D. McDaniel, and Manos Antonakakis. Domain-z: 28 registrations later measuring the exploitation of residual trust in domains. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'16)*, pp. 691–706, 2016.

[25] Google play asi program. https://developer.android.com/google/play/asi.html, 2017.

[26] Ryan Stevens, Jonathan Crussell, and Hao Chen. On the origin of mobile apps: Network provenance for android applications. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY'16)*, pp. 160–171, 2016.

[27] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu. The achilles heel of oauth: A multi-platform study of oauth-based authentication. In *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC'16)*, pp. 167–176, 2016.

[28] Certificatepinning. https://developer.android.com/training/articles/security-configCertificatePinning, 2018.

[29] Justin Del Vecchio, Feng Shen, Kenny M. Yee, Boyu Wang, Steven Y. Ko, and Lukasz Ziarek. String analysis of android applications (N). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, pp. 680–685, 2015.

[30] Yi Chen, Wei You, Yeonjoon Lee, Kai Chen, XiaoFeng Wang, and Wei Zou. Mass discovery of android traffic imprints through instantiated partial execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pp. 815–828, 2017.

[31] Farsight passive dns. https://www.farsightsecurity.com/solutions/dnsdb/, 2017.

[32] The menlo report: Ethical principles guiding information and communication technology research. https://www.caida.org/publications/papers/2012/menlo$_report_actual_formatted/$, 2012.

[33] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard A. Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security (CCS'09)*, pp. 635–647, 2009.

[34] Sumayah A. Alrwais, Kan Yuan, Eihal Alowaisheq, Zhou Li, and XiaoFeng Wang. Understanding the dark side of domain parking. In *Proceedings of the 23rd USENIX Security Symposium*, pp. 207–222, 2014.

[35] Shuang Hao, Matthew Thomas, Vern Paxson, Nick Feamster, Christian Kreibich, Chris Grier, and Scott Hollenbeck. Understanding the domain registration behavior of spammers. In *Proceedings of the 2013 Internet Measurement Conference (IMC'13)*, pp. 63–76, 2013.

[36] Shuang Hao, Alex Kantchelian, Brad Miller, Vern Paxson, and Nick Feamster. PREDATOR: proactive recognition and elimination of domain abuse at time-of-registration. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pp. 1568–1579, 2016.

[37] Tobias Lauinger, Abdelberi Chaabane, Ahmet Salih Buyukkayhan, Kaan Onarlioglu, and William Robertson. Game of registrars: An empirical analysis of post-expiration domain name takeovers. In *Proceedings of the 26th USENIX Security Symposium*, pp. 865–880, 2017.

[38] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pp. 356–367, 2016.

[39] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. Libd: scalable and precise third-party library detection in android markets. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*, pp. 335–346, 2017.

[40] Benjamin Andow, Adwait Nadkarni, Blake Bassett, William Enck, and Tao Xie. A study of grayware on google play. In *Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW'16)*, pp. 224–233, 2016.

[41] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'16)*, pp. 357–376, 2016.