Android Device Risk Assessment Tool:

Using Common Permissions to Identify Applications Used in Intimate Partner Violence

Kathryn Reardon

November 24, 2020

Abstract

* Write after finishing report
* 120 – 500 words, or 1-2 paragraphs
  + 25% on purpose/importance of research (Introduction)
  + 25% on what I did (Methods)
  + 35% on what I found (Results/Evaluation)
  + 15% on implications of research (Discussion)

1 Introduction

Intimate partner violence (IPV) is considered a public health concern in the United States by the CDC. This fact can be shocking for the uninformed whose idea of a public health concern is heart disease or a viral pandemic, but the four abusive behaviors of physical violence, sexual violence, stalking, and psychological aggression can lead to injury and death as assuredly as any disease [1]. The statistics differ slightly depending on the source, but approximately 1 in 4 women and 1 in 10 men have reported experiencing some form of IPV. One method that abusers use to control, manipulate, and harm their victims is surveillance spyware installed on their victim’s devices. This topic has only recently been researched, but it is a security issue of great importance because the abusers do not have to be technologically savvy to employ their attacks. Additionally, many victims are uninformed about the abuser’s tech capabilities and do not know how to deal with it even once the surveillance is suspected or discovered.

This project aims to continue the burgeoning research and security solutions begun by students and faculty primarily at Cornell Tech and New York University [2, 3, 4, 5, 6, 7]. The work done by Sam Havron et al. in creating a clinical computer security procedure and IPV Spyware Discovery tool (ISDi) is the major inspiration for the project [5]. Further work by Kevin Roundy et al. also provided ideas during the research phase [7]. I had several goals for the project, the first of which was to improve ISDi’s efficacy with a sort of “signature”-based spyware detection, as ISDi relies on blacklisting. The second goal was to obtain the permissions of known Android spyware applications and analyze them for commonalities in a guilt-by-association approach where any permission that was frequently used in spyware was more likely to indicate that an unknown app was also spyware. The third goal was to inspect victim devices without arousing suspicion of the attacker. The fourth goal was to programmatically assess the likelihood of an app being spyware with accuracy, as false negatives can be dangerous for the client (victim) while false positives obfuscate the danger. The final goal was to do everything in a way that is easy for a client to understand, leading to informed decision making regarding their device.

To accomplish these goals, I have designed three separate pieces, two support tools and a graphic user interface to display the findings. The first tool, AnalyzeAndroidPermissions, is Java code used to read the permissions from the AndroidManifest.xml files of a group of sample applications. It categorizes and sorts them according to Android protection level, number of times used, and whether they are present in spyware or popular, non-spyware apps. The second tool, Horoscope, is an Android application that on its face appears to be a simple daily horoscope. However, when the app is launched, it also gathers installation data of the apps on the device and saves them to a file to be used with the third tool. This final tool, AssessAppRisk, is a Java Swing application which lists every installed app with the data taken from the Horoscope app, and also displays a risk value decided by heuristic weights, a risk assessment, and descriptions of the permissions’ capabilities. AssessAppRisk includes a whitelist of 17 of the most popular apps on the Google Play Store, so the risk assessment ranges from whitelisted, very unlikely, unlikely, slightly likely, likely, and very likely to be spyware. If these tools were to be used in a real situation, it would follow the example of the field study introducing ISDi: a technician works with the client to run the tools on their device and go over the results with them.

Since IPV has evolved into a computer security problem, it is important to create a threat model that contextualizes the roles of attacker and victim and answers the questions of what the attacker knows, has access to, and can do. We must have a model both for the initial attack on the victim device and in the use of project’s tool. In the first case, these questions have been answered by the previous researchers [3]. As an aside, we use the terms attacker and victim here in the sense of an adversary model, though they align with the role of abuser and the target of abuse. “Victim” is not meant to be a slight or implication toward any person who is a target of or survivor of abuse. As for who the attacker and victim are, they tend to be intimate partners—spouses, boyfriends or girlfriends, exes, etc. There are other types of relationships, such as that of parent and child (where either party may play either role depending on age and circumstances) or even platonic friendships [8], but the relationships described in anecdotes by clients of IPV studies were only those of a romantic partner, ex-partner, and/or parent of their child(ren). The attacker and victim may live together, have lived together in the past, or never together at all. This is an important distinction for this threat model; while there are insider attacks wherein the attacker uses their organizational status to carry out an attack on that organization, it is easier to imagine a hacker thousands of miles away carrying out a remote attack. This is also not an attack on some faceless corporation, but one on a specific individual, and one whom the attacker knows very well.

What the attacker knows in this situation is both much and very little. On the social engineering side, the attacker may know many secrets that allow them to compromise the victim’s devices or accounts. Over the course of their relationship with the victim, they may come to know or compel the victim to tell them answers to security questions, like favorite color or birthday; they can watch or compel the victim to type passwords and PINs. On the other hand, it has been found that these attackers are not technically sophisticated, and Freed et al. go so far as to term them “UI-bound adversaries,” as they employ their attacks through a standard user interface with which they can be authenticated, or download the applications examined in this project to do their surveillance [3].

Another drastic difference from the typical adversary model is the attacker’s access to the victim device since most, if not all, assume the attacker’s target is someone else’s device [11]. One commonly reported scenario in cases of IPV is the device or its service is bought or paid for by the attacker. This gives the attacker control in innumerable ways, with such examples as the attacker confiscating or destroying the device, controlling associated digital accounts such as the mobile family plan or iCloud, and even manipulating relationships by giving a child a device with the intention to harass the adult target through it. Physical access to the device is key, especially since many of the spyware apps marketed towards these attackers require it.

Finally, what the attacker can do varies and has changed over time. In the past, an attacker could search for simple terms on the Google Play store like “track my girlfriend’s phone without them knowing” or “read SMS from another phone” and found many apps to choose from [4]. Following the warnings of security researchers, Google has removed many spyware apps from its store and filtered out IPV-related search terms, and it seems Android also has made changes to its APIs to make certain features of spyware apps unusable. Still, there were and still are apps that can be found from a Google search, and it is simple to disable a device’s protections, e.g., Google’s Play Protect, to install such off-store apps. One particularly nasty app called Cerberus boasts of uninstall protection, remote wipe, lock with password, blocking the power menu, and those are only the capabilities that prevent the victim from reclaiming their privacy. Apps claim they can track the device’s location, take pictures, record video and audio, forward text messages, read deleted messages, and practically any other type of privacy breach one can imagine. This makes them powerful and scary tools indeed. On the other hand, some apps are not as they appear. Some apps tested for this project did not work past an introductory screen, and others triggered anti-virus software which flagged the app as a phishing attack. The Zscaler research term confirmed another case when analyzing the code of the keylogger app SPYMIE, finding a hard-coded email address with a timer to send surveilled data every minute [12]. Additionally, while some apps are free to download, most can only be used after purchasing a subscription plan.

I had to evaluate a threat model when designing my project as well. Many aspects from the previous threat model remain. I kept many design considerations from Sam Havron et al. when they were creating ISDi. First we had to consider if the attacker would know that a spyware scanning tool was being used on the victim device, and the following consideration was what they would do upon learning of the tool’s use or the victim’s participation in a security clinic. I will discuss the implications of these considerations on both projects in later sections.

Given that the main goal of my project is to identify Android spyware used in IPV, I would evaluate my project’s success by the accuracy with which it does so. I would like to see AssessAppRisk flag 75% of known spyware apps as some degree of likely to be spyware while flagging only 25% of non-spyware apps (false positives). I will show an evaluation in a following section.

2 Background and Related Work

The groundwork for this project was laid by an interdisciplinary group of researchers at Cornell Tech, Cornell University, and New York University in a series of studies beginning in 2017 and continuing today. The first study largely identifies IPV as a security issue through interviews with 40 IPV professionals and nine focus groups with 32 survivors of IPV. These interviews revealed how abusers use technology, what clients and professionals understand about said technology, how professionals advise clients about technology, and how the law understands technology used in IPV [2]. The next paper was another qualitative study with 89 participants to detail exactly how abusers use technology in IPV, which discovered technologically unsophisticated methods, such as social engineering and downloaded applications. Most importantly, they began to consider how to mitigate these types of attacks [3]. This led to an investigation of spyware used in intimate partner surveillance which revealed both spyware and what they termed dual-use apps which have legitimate uses but can be exploited by abusers, such as Find My Friends or anti-theft applications. In this study, they also found that existing anti-virus and anti-spyware tools did not consistently catch dual-use applications [4].

The next study, mentioned earlier, produced a consultation service for IPV victims. In this service, a trained technologist worked with the client and an IPV professional to answer a standardized technology assessment questionnaire, create a diagram summarizing the client’s digital assets, manually check the client’s device for security configurations, and scan the device using ISDi [5]. ISDi is a Python application that does not require installation on the client’s device. This is to avoid notifying the abuser that the device is being investigated, since some spyware applications keep track of application installations or take screenshots while the device is being used. Instead, the technologist uses a USB connection to a laptop where ISDi is run in a browser. The researchers tested a USB connection on devices with six more capable spyware apps installed and were reasonably confident that the apps would not pick up on ISDi. ISDi’s major limitation is that its spyware detection is rather simple, using a blacklist of application names that were found through machine learning [4].

The next paper published in 2019 examined the findings of the clinical computer security approach and was more focused on sociology than computer science, reiterating some of the ideas from previous papers about the security needs and understandings of IPV victims [6]. The last study used the spyware applications unearthed by Chatterjee et al. as a seed set to discover “creepware,” apps similar to the surveillance-based spyware previously discussed but used more generally for interpersonal attacks. This paper is as important to this project as the paper concerning ISDi as it provided a method for spyware detection beyond blacklisting. The researchers developed an algorithm called CreepRank which uses the principle of guilt-by-association to identify applications as creepware [7].

3 Dataset Description and Properties

The first step in researching the permissions used in IPV-related applications was to obtain a representative sample of spyware applications, as well as “safe” popular applications for comparison. Most of the spyware apps (listed in Table II of Appendix section A) were referenced in previous work, while the popular apps (listed in Table III) were chosen either because they were suggested on the Google Play store, deemed the most downloaded, or reported on in the media recently. It is necessary to examine the permissions of both spyware and popular applications because there is bound to be overlap. For example, one of the features nearly every app used for IPS has is location tracking; it goes without saying that apps like Lyft and Google Maps also require location. Then there is the matter of apps which realistically should not need your location but ask for it anyway, like Facebook and TikTok. Therefore, to reach the goal of successfully identifying spyware apps while not falsely identifying popular apps, both sets of permissions needed to be analyzed.

I chose only applications available on Android devices for several reasons: it is easier to get Android application information, I was anticipating using Android Studio for part of the project, and I could readily install applications on a physical Android device via USB and Windows 10. When initially researching the possibilities for this project, I found lots of discussion about programmatically mining the data of installed applications on an Android device and little about iOS devices. What little I found implied that it was impossible. Since I had a little experience with Android Studio, I wanted to use it for the Horoscope app; additionally, although iOS apps can be written in Java, Java is better supported for Android apps, and I have most of my programming experience in that language. Lastly, when testing USB connections on my development PC, I successfully connected an Android phone in a plug-and-play manner but no success with an iPhone. I also felt that developing for iOS devices was better suited as a future work endeavor given the confines of the project timeline.

The program AnalyzeAndroidPermissions produced the data needed to develop the AssessAppRisk program. By design, every Android app must include an *AndroidManifest.xml* file. This file describes essential information about the app, including its package name and permissions that the app will request [15]. The package name is important because some spyware apps hide their nature by pretending to be system services named, for example, “Update manager” or “Sync Service.” A user granting permissions is what allows an app to access otherwise protected parts of the system, such as the camera, contacts, or external media. When I found a small sample of applications, I downloaded their APK files to my computer, APK being how Android packages applications. These files are not readable without reverse engineering, so I used the open-source Apktool to extract the AndroidManifest files and save them in a resources folder for AnalyzeAndroidPermissions [16]. I later used my own Horoscope app to get data from more apps that were installed on an emulated device, so the resource files are mixture of extracted XML and handwritten text files.

To begin, the program reads all of the files in the spyware or popular folder. It searches for the prefix *<uses-permission android:name=*, which is found before every permission, and adds it to a new line in a text file (*spywarePermissions.txt* or *popularPermissions.txt*). Repeated permissions between files are included because I want to see which permissions are requested most frequently. With all of the permissions consolidated, I first referenced the Android permissions documentation to determine if a permission was install-time (normal and signature) or runtime (dangerous) [17]. These protection levels indicate the scope of data accessed and actions performed by the app, so I hoped different level would correlate with the permissions used in spyware or popular apps. Install-time permissions are automatically granted when the user installs the app, which means they are considered minimally risky for other apps, the system, or the user, so many permissions are considered normal. The system grants signature permissions only if the requesting application is signed with the same certificate as the application that declared the permission, and some signature permissions are not meant for third party use. Dangerous runtime permissions, however, must be explicitly granted by the user when first opening the application. They allow an application to access a higher level of private user data or control over the device. Unfortunately, Android’s documentation is not comprehensive. It was difficult to find information about permissions that were deprecated or removed in previous versions of the API, and these difficult-to-find permissions were categorized using other sources [18]. I labeled permissions that are not in the current API as removed. Permissions that seem to have similar purposes but which I could not find clear documentation for were classified as other, while some permissions are developer-specified, which I classified as unknown by default since I cannot predict the naming scheme or usage.

The next step was to place each permission in a hash map with its name as the key and the number of times it appeared in applications and its protection level as the value, then sort the map by protection level and count. This was done separately for the spyware and popular permissions. The final function of AnalyzeAndroidPermissions was to create lists of permissions unique to spyware, unique to popular, and shared between the two. These lists also included the permission’s protection level and the percent of applications which used it. I then copied this data to Microsoft Excel to do some additional sorting and see the data side by side, which helped me to better notice patterns or interesting outliers. This data is in section B of the Appendix.

4 Using Guilt-by-Association

As mentioned earlier, the CreepRank algorithm developed by Roundy et al. informed the design of AssessAppRisk’s risk assessment algorithm. CreepRank uses graph mining to compute scores using maximum a posteriori estimation so that the higher the score, the more the app is associated with known creepware [7]. The algorithm takes a seed set of apps discovered by Chatterjee et al. as input and a dataset of installed applications and outputs the CreepRank for each application. AssessAppRisk’s algorithm is similar, taking the set of permissions discovered by AnalyzeAndroidPermissions and a set installed applications and outputs a rank for each application. The CreepRank algorithm is much more sophisticated, however, as the researchers used bipartite graphs to represents apps and devices as nodes and edges to represent installation. They then estimated the probability *p* that a particular app appears on a spyware-infected device using a binomial distribution and maximum likelihood estimation; they later incorporated maximum a posteriori probability estimates when the MLE method proved to suffer from high false positive rates.

Although I would have preferred AssessAppRisk to have a mathematically-driven algorithm, I instead decided on a heuristic approach based on what I saw in the permission analysis. The first decision was that permissions of different protection levels would be more or less indicative of spyware use. This is obvious in the case of dangerous permissions; indeed, you can see in Table V and Table VI that the spyware apps use more unique dangerous permissions than the popular apps, and in Table IV that the spyware apps use more dangerous apps at a slightly higher frequency than the popular apps. I ignored the normal permissions in the assessment because of the high overlap between spyware and popular apps and the fact that they have little use for attackers.

I made two other decisions based on intuition. The first was noticing that some spyware applications used five unique removed permissions while some popular applications used only two. The difference is small, but I considered how the spyware apps are used in practice: an attacker installs the app on the victim’s device while they have physical access to it, and they may not have access to that device again. This could mean the spyware application, while running on the victim’s device, is never or rarely updated. It is also possible that the developers do not update the app frequently, unlike big-name development companies like Facebook and Twitter. Both of these theories could mean that spyware contains removed permissions at a higher rate. The second decision I made was about signature permissions. I noticed that a substantial number of signature permissions were used by spyware apps, with only three permissions shared between spyware and popular apps and popular apps using only two unique permissions. This imbalance combined with the fact that many signature applications are not to be used by third parties made signature permissions a red flag for spyware.

Based on these observations, I separated the permissions into nine groups and assigned a different weight.

|  |  |
| --- | --- |
| Category | Weight |
| Very Dangerous | 3 + frequency |
| Moderately Dangerous | 2 + frequency |
| Less Dangerous | 1 + frequency |
| Only Spyware Dangerous | 1.5 + frequency |
| Only Popular Dangerous | 1 |
| Shared Signature | 1 + frequency |
| Only Spyware Signature | 3 + frequency |
| Only Popular Signature | 1 |
| Removed | 0.3 |

Table I: Heuristic values for risk assessment.

The frequency value is the fraction of spyware apps that use a particular permission. Very Dangerous includes dangerous permissions that were included in a large majority (over 70%) of spyware applications, with the reasoning that if most spyware apps used them, they must be important to the basic surveillance functionality of the apps. Moderately Dangerous includes dangerous permissions used by fewer than 70% and more than 30% of the spyware apps, while Less Dangerous includes the remaining dangerous permissions. The rest of the categories are as they are named. When the risk value is calculated for an individual app, the total number of permissions is also accounted for. As you can imagine from such a simple weight scheme, an app with many weighted permissions will have a higher score than an app with only a few, and some apps in the sample only used around 10 total permissions. This does not exclude them from being spyware, so I added a modifier to the risk score. This modifier is the ratio of weighted permissions used in the application to the total number of permissions. If this ratio is greater than 0.4, then the risk score is increased by 1 + ratio, which helps to normalize the risk score.

5 The Risk Assessment Tool

As mentioned earlier, the Horoscope application and AssessAppRisk program are meant to be used to investigate the victim device with the help of a technician to run and interpret the programs. The technician would have Android Studio installed on a laptop which also has the project files for Horoscope and AssessAppRisk. The technician would put the victim device into debugging mode and connect it to the laptop via USB. The Horoscope application can then be installed on the device through Android Studio. After obtaining the application data from the device, the technician saves the output file to the resources folder in AssessAppRisk’s source folder. After running AssessAppRisk, the technician and client can view the results together with the technician providing explanation and advice. The tools are not meant to be used alone by victims of IPV for several reasons. The first is releasing either the Horoscope application or AssessAppRisk program to the public would make abusers equally aware of its use as survivors, and knowing that their attacks have been thwarted could cause them to change their methods or escalate their violent behavior. Another reason is that the clients surveyed in earlier studies were not typically knowledgeable about technology and may have trouble understanding the results. Finally, the way this project is designed prohibits the average non-programmer from using it; this is because the project is experimental, not because this is the best design.

The Horoscope application is an Android application with two parts: the front-facing daily horoscope and the backend application listing function. Horoscope uses Android’s PackageManager class to retrieve information about the application packages installed on the device including the application name, package name, installation date, last modified date, and permissions [9]. It also filters system applications and Horoscope itself since we are only interested in applications that were user-installed. Horoscope writes to file the application information, and the file is accessible in Android Studio by clicking *View > Tool Windows > Device File Explorer > data/data/net.sleepinginthevoid.horoscope > appdata.txt*. This file can then be saved to AssessAppRisk’s resources folder.

Horoscope is designed to meet the goal of inspecting the victim device without alerting the abuser. Unlike ISDi, this approach does involve installing an application, so I have taken some efforts make up for this. I feel that if the application looks innocuous, it is less likely to arouse suspicion. If one were to search the Google Play or iTunes store, they would find many similarly titled and themed apps, and it is easy to imagine anyone downloading such an app for fun or out of boredom. The app also does what it says—it displays a daily horoscope for each astrological sign, which makes it more believably innocent. The text is hardcoded as Horoscope can be uninstalled as soon as it runs; it is not meant to stay on the device longer than necessary. On the backend, the package name keeps with the horoscope theme; unlike the spyware applications which fake their name and reveal their true nature with their package name, Horoscope maintains its disguise rather than being named something like “applist.” Finally, the actual application listing and risk assessment happens off of the victim device. This is so any spyware installed on the device does not report the assessment, and even if an attacker were to reverse engineer Horoscope’s code, they would not know the purpose of the application data being gathered.

The final piece of the project is the main feature, AssessAppRisk. Since *appdata.txt* must be saved to the project files, the program is run from an IDE. When the program is run, it creates a Java Swing application window with four panels. The top left panel displays a list of installed applications and their package name. Clicking on an item changes the appearance of the other three panels. The bottom left panel displays the risk score, first installation date, last modified date, and permissions. The bottom right panel again lists the permissions, but with details about their protection level and a brief description of their use (mostly taken from the Android documentation). The top right panel displays a qualitative risk assessment with a description of what that assessment could mean.

The program works by first creating a hash map to store the permission weights described in section 4. It also reads from the *whitelist.txt* file in the resources folder, which contains 19 very popular applications. This whitelist is intended to lower the number of false positives AssessAppRisk returns. When the user clicks on an application in the list, the program calculates the risk score and returns it along with the other information displayed in the bottom left panel. The risk score is calculated by comparing the application’s permissions to the weighted permissions in the permission weight map, adding the values for all weighted permissions, and adjusting the total score by the ratio discussed earlier.

The risk assessment follows another simple heuristic evaluation. Any whitelisted application is noted as such, while any other app may be rated as very unlikely, unlikely, slightly likely, likely, or very likely to be spyware, which is decided on the range in which the app’s risk score falls. The ranges are 0, greater than 0 and less than 20, greater than or equal to 20 and less than 50, greater than or equal to 50 and less than 70, and greater than 70. These ranges were determined by observing the risk scores from the sample applications and how they roughly broke down among the spyware and popular applications. Obviously the higher the risk score, the more likely the application is to be spyware. If an app were to use every single weight permission, its score would be 129.05; this is extremely unlikely since many of the permissions are used in only one or two applications, but it does show the relative severity of some applications over others.

6 Evaluating Efficacy

* Show testing examples
* Evaluate results—satisfactory? Does it meet established goals? Mixed results?
* Thoughts about results

7 Discussion

* Did I achieve my goals? Which ones?
* Limitations/room for improvement
* Implications of research
* Future work/context of project beyond this experimental stage

8 Conclusion

* Summarize accomplishments

References

Appendix

Code

Screenshots

REFERENCES

1. CDC. 2020. Intimate Partner Violence. Retrieved November 17, 2020 from

https://www.cdc.gov/violenceprevention/intimatepartnerviolence/index.html.

1. Diana Freed, Jackeline Palmer, Diana Elizabeth Minchala, Karen Levy, Thomas

Ristenpart, and Nicola Dell. 2017. Digital technologies and intimate partner violence: A qualitative analysis with multiple stakeholders. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW, Article 46 (November 2017), 22 pages. DOI:https://doi.org/10.1145/3134681

1. Diana Freed, Jackeline Palmer, Diana Minchala, Karen Levy, Thomas Ristenpart, and

Nicola Dell. 2018. “A stalker's paradise”: How intimate partner abusers exploit technology. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (*CHI '18*). April 21 – 26, 2018, Montréal, Canada. Association for Computing Machinery, New York, NY, USA, Paper 667, 1–13. DOI:https://doi.org/10.1145/3173574.3174241

1. Rahul Chatterjee et al. 2018. The spyware used in intimate partner violence. In

*Proceedings of the 2018 IEEE Symposium on Security and Privacy* (*SP*), May 21 – 23, 2018, San Francisco, California, USA. 441-458. DOI:https://doi.ieeecomputersociety.org/ 10.1109/SP.2018.00061

1. Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas

Ristenpart. 2019. Clinical computer security for victims of intimate partner violence. In *Proceedings of the 28th USENIX Security Symposium* (*Security 2019*). August 14 – 16, 2019, Santa Clara, California, USA. 105-122. DOI:https://dl.acm.org/doi/10.5555/3361338.3361347

1. Diana Freed, Sam Havron, Emily Tseng, Andrea Gallardo, Rahul Chatterjee, Thomas

Ristenpart, and Nicola Dell. 2019. “Is my phone hacked?” Analyzing clinical computer security interventions with survivors of intimate partner violence. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW, Article 202 (November 2019), 24 pages. DOI:https://doi.org/10.1145/3359304

1. Kevin A. Roundy, Paula Barmaimon Mendelberg, Nicola Dell, Damon McCoy, Daniel

Nissani, Thomas Ristenpart, and Acar Tamersoy. 2020. The many kinds of creepware used for interpersonal attacks. In *Proceedings of the 41st IEEE Symposium on Security and Privacy* (*Oakland 2020*). May 18 – 20, 2020, Online. 626-643. DOI: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00069

1. Karen Levy, Bruce Schneier, 2020. Privacy threats in intimate relationships. *Journal of*

*Cybersecurity* 6, 1 (May 2020), 13 pages. DOI:https://doi.org/10.1093/cybsec/tyaa006.

1. Nithya Vasudevan. How to Get List of Installed Apps in Android. Retrieved November

17, 2020 from https://theopentutorials.com/tutorials/android/listview/how-to-get-list-of-installed-apps-in-android.

1. Oracle. How to Write a List Selection Listener. Retrieved November 17, 2020 from

https://docs.oracle.com/javase/tutorial/uiswing/events/listselectionlistener.html.

1. Quang Do, Ben Martini, and Kim-Kwang Raymond Choo. 2019. The role of the

adversary in applied security research. *Computers & Security*, 81 (March 2019), 156-181. DOI:https://doi.org/10.1016/j.cose.2018.12.002.

1. Shivang Desai. 2018. Why You Shouldn’t Trust “Safe” Spying Apps! Retrieved November 17, 2020 from https://www.zscaler.com/blogs/security-research/why-you- shouldnt-trust-safe-spying-apps.
2. Computer Security and Privacy for Survivors of Intimate Partner Violence. Retrieved

November 17, 2020 from https://www.ipvtechresearch.org/.

1. Shivang Desai. 2019. A New Wave of Stalkerware Apps. Retrieved November 21, 2020 from https://www.zscaler.com/blogs/security-research/new-wave-stalkerware-apps.
2. Android. App Manifest Overview. Retrieved November 22, 2020 from https://developer.android.com/guide/topics/manifest/manifest-intro.
3. Connor Tumbleson. 2019. Apktool. Retrieved November 22, 2020 from

https://ibotpeaches.github.io/Apktool/.

1. Android. Permissions on Android. Retrieved November 22, 2020 from https://developer.android.com/guide/topics/manifest/permission-element.
2. Android Open Source Project. 2020. AndroidManifest.xml. Retrieved November 22, 2020 from https://github.com/aosp- mirror/platform\_frameworks\_base/blob/master/core/res/AndroidManifest.xml.

APPENDIX

1. *Sampled Applications*

|  |  |
| --- | --- |
| Application Name | Package Name |
| AllTracker Anti-Theft [14] | city.russ.alltrackercorp |
| Update manager [14] | com.ibm.fb |
| Cell Tracker Kids | us.cell.tracker.kids.locator |
| Cerberus [4] | com.lsdroid.cerberuss |
| Clone WhatsWeb Pro [7] | clone.whatsapp.pro |
| Control by SMS [7] | smartmob.com.controller |
| FamiSafe | com.wondershare.famisafe |
| Girlfriend Cell Tracker [7] | com.omrup.cell.tracker |
| Highster Mobile | com.highstermobile.main |
| Hoverwatch | com.android.core.mntg |
| iKeyMonitor [4] | com.sec.android.internet.im.service.im20190419 |
| IP Webcam [7] | com.pas.webcam |
| iSpyoo [14] | com.systemservice |
| Mrecorder [7] | com.mobileservice.sync |
| Screen Recorder [7] | net.example.hatiboy.gpcapture |
| SMS Forwarder [7] | cz.psencik.smsforwarder |
| Spy GPS SMS Call Controller [7] | com.dspark.phone.modefind |
| Spy to Mobile [7] | com.spy2mobile.light |
| TrackView [4] | com.trackview |
| Unseen – No Last Seen [7] | com.tda.unseen |

Table II: Set of IPS-related apps (more simply called spyware) used for permission analysis. Applications without a reference were found by searching online for IPS-related terms.

|  |  |
| --- | --- |
| Application Name | Package Name |
| Amazon | com.amazon.mShop.android.shopping |
| Among Us | com.innersloth.spacemafia |
| Cash App | com.squareup.cash |
| Facebook | com.facebook.katana |
| Gmail | com.google.android.gm |
| Google Maps | com.google.android.apps.maps |
| Instagram | com.instagram.android |
| Lyft | me.lyft.android |
| Facebook Messenger | com.facebook.orca |
| Netflix | com.netflix.mediaclient |
| Pokémon Go | com.nianticlabs.pokemongo |
| Seamless | com.seamlessweb.android.view |
| Snapchat | com.snapchat.android |
| Spotify | com.spotify.music |
| TikTok | com.zhiliaoapp.musically |
| Twitter | com.twitter.android |
| Uber | com.ubercab |
| Weather Channel | com.weather.Weather |
| WhatsApp | com.whatsapp |
| YouTube | com.google.android.youtube |

Table III: Set of non-IPS-related apps (more simply called popular) used for permission analysis.

1. *Sampled Permissions*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Shared | | Spyware | Popular | Combined |
| dangerous | android.permission.WRITE\_EXTERNAL\_STORAGE | 0.85 | 0.8 | 0.83 |
| dangerous | android.permission.ACCESS\_FINE\_LOCATION | 0.75 | 0.7 | 0.73 |
| dangerous | android.permission.READ\_EXTERNAL\_STORAGE | 0.75 | 0.7 | 0.73 |
| dangerous | android.permission.ACCESS\_COARSE\_LOCATION | 0.7 | 0.6 | 0.65 |
| dangerous | android.permission.READ\_CONTACTS | 0.7 | 0.65 | 0.68 |
| dangerous | android.permission.READ\_PHONE\_STATE | 0.7 | 0.65 | 0.68 |
| dangerous | android.permission.RECORD\_AUDIO | 0.55 | 0.5 | 0.53 |
| dangerous | android.permission.CAMERA | 0.5 | 0.7 | 0.6 |
| dangerous | android.permission.READ\_SMS | 0.5 | 0.1 | 0.3 |
| dangerous | android.permission.RECEIVE\_SMS | 0.5 | 0.15 | 0.33 |
| dangerous | android.permission.GET\_ACCOUNTS | 0.45 | 0.7 | 0.57 |
| dangerous | android.permission.CALL\_PHONE | 0.35 | 0.25 | 0.3 |
| dangerous | android.permission.ACCESS\_BACKGROUND\_LOCATION | 0.3 | 0.15 | 0.23 |
| dangerous | android.permission.ACCESS\_MEDIA\_LOCATION | 0.25 | 0.15 | 0.2 |
| dangerous | android.permission.READ\_CALENDAR | 0.25 | 0.2 | 0.23 |
| dangerous | android.permission.SEND\_SMS | 0.25 | 0.15 | 0.2 |
| dangerous | android.permission.RECEIVE\_MMS | 0.1 | 0.05 | 0.08 |
| dangerous | android.permission.WRITE\_CONTACTS | 0.05 | 0.2 | 0.13 |
| normal | android.permission.ACCESS\_NETWORK\_STATE | 0.95 | 1 | 0.98 |
| normal | android.permission.INTERNET | 0.95 | 1 | 0.98 |
| normal | android.permission.WAKE\_LOCK | 0.95 | 1 | 0.98 |
| normal | android.permission.ACCESS\_WIFI\_STATE | 0.75 | 0.8 | 0.78 |
| normal | android.permission.RECEIVE\_BOOT\_COMPLETED | 0.7 | 0.85 | 0.78 |
| normal | android.permission.CHANGE\_WIFI\_STATE | 0.45 | 0.4 | 0.43 |
| normal | android.permission.MODIFY\_AUDIO\_SETTINGS | 0.4 | 0.5 | 0.45 |
| normal | android.permission.VIBRATE | 0.35 | 0.85 | 0.6 |
| normal | android.permission.BLUETOOTH | 0.3 | 0.55 | 0.43 |
| normal | android.permission.FOREGROUND\_SERVICE | 0.25 | 0.7 | 0.48 |
| normal | android.permission.CHANGE\_NETWORK\_STATE | 0.2 | 0.25 | 0.23 |
| normal | android.permission.GET\_TASKS | 0.2 | 0.15 | 0.18 |
| normal | android.permission.BLUETOOTH\_ADMIN | 0.1 | 0.45 | 0.28 |
| normal | com.android.launcher.permission.INSTALL\_SHORTCUT | 0.1 | 0.35 | 0.23 |
| normal | com.android.launcher.permission.UNINSTALL\_SHORTCUT | 0.1 | 0.15 | 0.13 |
| normal | android.permission.BROADCAST\_STICKY | 0.05 | 0.15 | 0.1 |
| normal | android.permission.CHANGE\_WIFI\_MULTICAST\_STATE | 0.05 | 0.2 | 0.13 |
| normal | android.permission.DISABLE\_KEYGUARD | 0.05 | 0.05 | 0.05 |
| normal | android.permission.GET\_PACKAGE\_SIZE | 0.05 | 0.05 | 0.05 |
| normal | android.permission.NFC | 0.05 | 0.4 | 0.23 |
| normal | android.permission.READ\_SYNC\_SETTINGS | 0.05 | 0.25 | 0.15 |
| normal | android.permission.READ\_SYNC\_STATS | 0.05 | 0.1 | 0.08 |
|  | Shared | Spyware | Popular | Combined |
| normal | android.permission.USE\_BIOMETRIC | 0.05 | 0.35 | 0.2 |
| normal | android.permission.USE\_FINGERPRINT | 0.05 | 0.45 | 0.25 |
| normal | android.permission.USE\_FULL\_SCREEN\_INTENT | 0.05 | 0.2 | 0.13 |
| other | com.google.android.c2dm.permission.RECEIVE | 0.65 | 0.9 | 0.78 |
| other | com.google.android.finsky.permission.BIND\_GET\_INSTALL\_REFERRER\_SERVICE | 0.4 | 0.6 | 0.5 |
| other | com.android.vending.BILLING | 0.3 | 0.35 | 0.33 |
| other | com.google.android.gms.permission.ACTIVITY\_RECOGNITION | 0.1 | 0.2 | 0.15 |
| other | com.android.launcher.permission.READ\_SETTINGS | 0.05 | 0.1 | 0.08 |
| other | com.google.android.providers.gsf.permission.READ\_GSERVICES | 0.05 | 0.55 | 0.3 |
| removed | android.permission.READ\_PROFILE | 0.15 | 0.4 | 0.28 |
| removed | android.permission.USE\_CREDENTIALS | 0.15 | 0.5 | 0.33 |
| removed | android.permission.FLASHLIGHT | 0.1 | 0.15 | 0.13 |
| removed | android.permission.WRITE\_SMS | 0.1 | 0.05 | 0.08 |
| removed | android.permission.AUTHENTICATE\_ACCOUNTS | 0.05 | 0.35 | 0.2 |
| removed | android.permission.MANAGE\_ACCOUNTS | 0.05 | 0.45 | 0.25 |
| signature | android.permission.SYSTEM\_ALERT\_WINDOW | 0.45 | 0.25 | 0.35 |
| signature | android.permission.WRITE\_SETTINGS | 0.3 | 0.05 | 0.18 |
| signature | android.permission.REQUEST\_INSTALL\_PACKAGES | 0.1 | 0.1 | 0.1 |

Table IV: Shared permissions (unknown omitted) with fraction of apps that included them.

|  |  |  |
| --- | --- | --- |
| Spyware Only | | |
| dangerous | android.permission.READ\_CALL\_LOG | 0.5 |
| dangerous | android.permission.PROCESS\_OUTGOING\_CALLS | 0.45 |
| dangerous | android.permission.ACCESS\_SUPERUSER | 0.15 |
| dangerous | android.permission.READ\_PHONE\_NUMBERS | 0.1 |
| dangerous | android.permission.WRITE\_CALL\_LOG | 0.1 |
| dangerous | android.app.action.DEVICE\_ADMIN\_ENABLED | 0.05 |
| normal | android.permission.REQUEST\_IGNORE\_BATTERY\_OPTIMIZATIONS | 0.35 |
| normal | android.permission.REQUEST\_DELETE\_PACKAGES | 0.2 |
| normal | android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS | 0.1 |
| normal | android.permission.RESTART\_PACKAGES | 0.1 |
| normal | android.permission.ACCESS\_NOTIFICATION\_POLICY | 0.05 |
| normal | android.permission.KILL\_BACKGROUND\_PROCESSES | 0.05 |
| normal | android.permission.PERSISTENT\_ACTIVITY | 0.05 |
| normal | android.permission.SET\_ALARM | 0.05 |
| removed | com.android.browser.permission.READ\_HISTORY\_BOOKMARKS | 0.25 |
| removed | com.android.browser.permission.WRITE\_HISTORY\_BOOKMARKS | 0.1 |
| removed | android.permission.READ\_HISTORY\_BOOKMARKS | 0.05 |
| removed | android.permission.READ\_USER\_DICTIONARY | 0.05 |
| removed | android.permission.WRITE\_USER\_DICTIONARY | 0.05 |
| signature | android.permission.PACKAGE\_USAGE\_STATS | 0.2 |
| signature | android.permission.READ\_PRIVILEGED\_PHONE\_STATE | 0.15 |
| signature | android.permission.INTERACT\_ACROSS\_USERS | 0.1 |
| signature | android.permission.INTERACT\_ACROSS\_USERS\_FULL | 0.1 |
| signature | android.permission.READ\_LOGS | 0.1 |
| signature | android.permission.WRITE\_SECURE\_SETTINGS | 0.1 |
| signature | android.app.action.BIND\_DEVICE\_ADMIN | 0.05 |
| signature | android.permission.BIND\_ACCESSIBILITY\_SERVICE | 0.05 |
| signature | android.permission.BIND\_NOTIFICATION\_LISTENER\_SERVICE | 0.05 |
| signature | android.permission.CAPTURE\_AUDIO\_OUTPUT | 0.05 |
| signature | android.permission.CHANGE\_CONFIGURATION | 0.05 |
| signature | android.permission.MANAGE\_DEVICE\_ADMINS | 0.05 |
| signature | android.permission.MANAGE\_USERS | 0.05 |
| signature | android.permission.MODIFY\_PHONE\_STATE | 0.05 |
| signature | android.permission.OVERRIDE\_WIFI\_CONFIG | 0.05 |
| signature | android.permission.REBOOT | 0.05 |
| signature | android.permission.STATUS\_BAR | 0.05 |
| signature | android.permission.TETHER\_PRIVILEGED | 0.05 |
| signature | android.permission.UPDATE\_APP\_OPS\_STATS | 0.05 |
| signature | com.android.launcher.permission.WRITE\_SETTINGS | 0.05 |

Table V: Permissions unique to spyware (unknown omitted).

|  |  |  |
| --- | --- | --- |
| Popular Only | | |
| dangerous | android.permission.ACTIVITY\_RECOGNITION | 0.15 |
| dangerous | android.permission.WRITE\_CALENDAR | 0.1 |
| normal | android.permission.WRITE\_SYNC\_SETTINGS | 0.3 |
| normal | android.permission.DOWNLOAD\_WITHOUT\_NOTIFICATION | 0.15 |
| normal | android.permission.MANAGE\_OWN\_CALLS | 0.1 |
| normal | android.permission.INSTALL\_SHORTCUT | 0.05 |
| normal | android.permission.REORDER\_TASKS | 0.05 |
| other | com.htc.launcher.permission.READ\_SETTINGS | 0.4 |
| other | com.htc.launcher.permission.UPDATE\_SHORTCUT | 0.4 |
| other | com.sec.android.provider.badge.permission.READ | 0.35 |
| other | com.sec.android.provider.badge.permission.WRITE | 0.35 |
| other | com.sonyericsson.home.permission.BROADCAST\_BADGE | 0.35 |
| other | com.huawei.android.launcher.permission.CHANGE\_BADGE | 0.3 |
| other | com.amazon.device.messaging.permission.RECEIVE | 0.25 |
| other | com.huawei.android.launcher.permission.READ\_SETTINGS | 0.2 |
| other | com.huawei.android.launcher.permission.WRITE\_SETTINGS | 0.2 |
| other | com.sonymobile.home.permission.PROVIDER\_INSERT\_BADGE | 0.2 |
| other | com.oppo.launcher.permission.READ\_SETTINGS | 0.15 |
| other | com.oppo.launcher.permission.WRITE\_SETTINGS | 0.15 |
| other | com.anddoes.launcher.permission.UPDATE\_COUNT | 0.1 |
| other | com.facebook.katana.provider.ACCESS | 0.1 |
| other | com.facebook.mlite.provider.ACCESS | 0.1 |
| other | com.facebook.orca.provider.ACCESS | 0.1 |
| other | com.facebook.permission.prod.FB\_APP\_COMMUNICATION | 0.1 |
| other | com.facebook.receiver.permission.ACCESS | 0.1 |
| other | com.majeur.launcher.permission.UPDATE\_BADGE | 0.1 |
| other | com.nokia.pushnotifications.permission.RECEIVE | 0.1 |
| other | me.everything.badger.permission.BADGE\_COUNT\_READ | 0.1 |
| other | me.everything.badger.permission.BADGE\_COUNT\_WRITE | 0.1 |
| removed | android.permission.SUBSCRIBED\_FEEDS\_READ | 0.05 |
| removed | android.permission.SUBSCRIBED\_FEEDS\_WRITE | 0.05 |
| signature | android.permission.BATTERY\_STATS | 0.05 |
| signature | android.permission.MANAGE\_DOCUMENTS | 0.05 |

Table VI: Permissions unique to popular apps (unknown omitted).