M-Perm: An Android Permissions Analysis Tool

Piper Chester, Cesar Perez and Daniel E. Krutz
Rochester Institute of Technology, Rochester, NY, USA
{pwc1203,cap7879,dxkvse}@rit.edu

ABSTRACT

Android Applications (apps) operate under a permissions-based system where access to specific APIs are restricted through the use of permissions. Unfortunately, there is no built-in verification system to ensure that apps do not request too many or too few permissions, which could lead to serious quality and/or privacy concerns. Apps requesting too many permissions create unnecessary vulnerabilities, leaving the potential for abuse by SDKs within the app or other malicious apps installed on the device.

With the introduction of Marshmallow 6.0 (M), the Android permission structure has substantially changed from requiring users to accept or reject all permissions when installing the app, to allowing developers to request permissions at run-time. Users also have finer control, gaining the ability to alter individual permissions after installation. Some of the motivating factors for switching to this new model include streamlining the app installation/update process and placing users more in control of the permissions their apps use.

In order to assist with the discovery of under and over-privileges within this new permission structure, we created a new detection tool, *M-Perm*, which can easily and effectively identify permission misuse in Android apps. M-Perm also identifies permission usage in apps including requested normal, dangerous, and 3rd party permissions. The tool, complete usage instructions, and screencast are available on our project website: http://www.m-perm.com

Categories and Subject Descriptors

D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification;

Keywords

Android, Smartphones, mobile phones, mobile security

1. INTRODUCTION

Android is the world's most popular mobile OS [14] with over 1.8 million apps available from Google Play alone [2]. A cornerstone of Android's security is its use of permissions. Android apps do not have any default permissions associated with them, so the developer must explicitly state the permissions needed by the app. The user must explicitly accept any requested 'dangerous' permissions, some of which include the ability to read SMS messages, record audio through the phone's microphone, and access the user's location [7].

Unfortunately, there is no built-in verification system to ensure that apps adhere to the *principle of least privilege* and do not request unnecessary permissions, which increases the app's attack surface and make it more susceptible to a variety of security and privacy related issues [18]. Although tools have been created to detect permission misuse and assist in the permission analysis of Android and its apps [9, 15], no known tools are tailored for the updated Android Marshmallow permission system.

Android Marshmallow 6.0 significantly changed the way Android apps ask the user to accept permissions. In the previous versions of Android, users granted an app's requested permissions at install time in an all or nothing fashion. If the user did not agree to all of the app's requested permissions, they would not be allowed to install the app. In the new permission model, users can decide to only allow the app access to specific permissions, and permissions may be requested at runtime. Users also have the ability to permit or revoke an app's permissions after it has been installed.

In order to detect permission misuse in apps using this new Android permission model, we have created a new publicly available tool *Marshmallow Permissions Analyzer* (M-Perm) which is available on our project website: **http://www.m-perm.com**. Our tool works by first reverse engineering the Android app (apk) file, and then comparing the permissions requested in the app's source code, against what is defined in the *AndroidManifest.xml* file. A configuration file allows the tool to easily adapt to future alterations in the Android permission structure. M-Perm also reports the app's requested normal, dangerous, and third party permissions.

We evaluated M-Perm in terms of effectiveness and efficiency against an app oracle with pre-defined instances of permissions misuse, and found that it is able to identify a high level of under and over-privileges in these apps. We also analyzed 50 existing Android apps and found that M-Perm is capable of finding a large number of misused permissions, and other permission related information in these apps.

The rest of the paper is organized as follows: Section 2 provides a background on the Android permission structure and Section 3 discusses M-Perm and how it works. Section 4 evaluates M-Perm against a set of 10 oracle apps, and then 50 apps collected from various sources. Section 5 describes how to get the tool and Section 6 provides a background on related works. Section 7 discusses limitations and future work to be conducted with M-Perm and Section 8 concludes our study.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. ANDROID PERMISSIONS

Android permissions are designed to protect the sensitive resources and information on the device. For example, in order for an app to use a device's camera, it must be granted access to the `Camera` permission. There are two primary steps in granting an app permission to use a specific resource. The developer must first tell the app to ask for the permission, which is done in the *AndroidManfiest.xml* file. Beginning with Marshmallow, the permissions must also be requested in the source code when the app needs access to that specific functionality. The final step in granting an app a permission is for the user to accept or reject this permission when using or installing the app. A primary goal of the Android permission model is to ensure that the user is only granting access to sensitive resources on their device which they are comfortable with.

Android apps created for versions up to, and including Lollipop (API 22) all require the user to accept or reject all permission requests at install time. If the user decided that they did not feel comfortable accepting any of the permissions, they would not be allowed to install, or even update the app if it requested any new permissions. Previous works have criticized the old model of requesting permissions at the beginning of the installation process in an all or nothing fashion for a variety of reasons, including the inability of a user to alter permission settings based on situational constraints such as who is using the device or their location [12,21]. According to Android, some of their primary goals for implementing the new permission model was to streamline the app installation and update process, while providing users more control over the permissions their apps used [6]. Dangerous permissions in the new model are also associated with groups. For example `SEND_SMS` and `READ_SMS` are part of the SMS group. If the user accepts any permission in the group, all other permissions in the group are automatically granted [7]. Android also allows developers to define their own custom, third party permissions [3,4].

The *principle of least privilege* is the concept of granting of the least amount of privileges to an application that it needs to properly function [24]. Granting extra privileges creates unnecessary security vulnerabilities by allowing malware to abuse these unused permissions, even in benign apps, along with making the app vulnerable to malicious SDKs [20]. These extra privileges also increase the app's attack surface [10,13].

Previous research has found that Android developers often mistakenly add unnecessary privileges in a counterproductive and futile attempt to make the app work properly, or due to confusion over the permission name [15]. In this study, we use the term *overprivilege* to describe a permission setting that grants more than what a developer needs for the task. Likewise, an *underprivilege* is a setting for which the app could fail because it was not given the proper permissions. The primary difference between requested permissions and overprivileges is that requested permissions are merely those that the app asks to use, and does not take into consideration if the app actually needs them or not.

3. M-PERM TOOL

M-Perm is comprised of two primary components, the reverse engineering module and the software analysis component. M-Perm was written in Python to make it accessible to numerous platforms, and to make it more maintainable by others since it is a very popular development language.

3.1 Reverse Engineering

The tool is composed of several different Python modules, with each component serving a distinct purpose. The Driver module first initiates the reverse engineering of the target Android APK, using several well known dependencies for APK reverse engineering which have been used in previous research [19, 30]: dex2jar[1], Apktool[2], and Procyon[3]. The reverse engineering project and scripts exist in a Git submodule within the main repository of the project. We chose to use a Git submodule so updates to the reverse engineering repository can easily be made, which is important since these reverse engineering submodules are regularly updated for routine updates and to handle alterations in Android.

3.2 App Analysis

Once the reverse engineering process is completed, the analysis of the app's source code can begin. The Analysis module performs static analysis on the disassembled APK and consists of parsing the application manifest, and then the disassembled source code. The source code analysis portion of M-Perm entails searching through every Java source file, and looking for references of the requested permission. For dangerous permissions, there must be a specific reference to use the permission within source. The analysis module also uses the provided configuration file from the user for grouping and identifying permissions. This configuration file is a text file with permissions and permission groups for the analysis module to ignore. This allows the user to isolate various permission groups during analysis. The analysis component concludes with producing a source report text file of all permission occurrences during analysis.

In the final stage of execution, the Report module examines the source report produced during analysis. This module handles most of the logic for examining which permissions have been requested by the manifest and which permissions are considered dangerous based on their group. Any permissions that occur in the source that were not requested in the manifest are considered underprivileged. Any permissions that didn't occur in the source that were requested in the manifest are considered overprivileged. M-Perm's output are stored in two .txt files in the *reports* directory. An overview of M-Perm's architecture is shown in Figure 1.

The Android permission structure regularly changes, and unfortunately these alterations often have a negative impact on a tool's ability to effectively examine an app's permissions. In order to accommodate change, M-Perm relies upon a configuration file *Permissions.py*, which holds the permission structure of the targeted Android OS. This file contains Android's dangerous and normal permissions, along with their defined groups. We plan on regularly updating this configuration file in our version control system to meet the changing structure of the Android permission system, although it should not be difficult for the typical developer to configure on their own as they desire.

3.3 Tool Output

Once completed, M-Perm's output is placed inside of two .txt files. The first is `analysis_<package_name>` which lists the normal, dangerous, 3rd party, and under & over privileges found in the app. The file also correlates the requested dangerous permissions with their associated permission groups, along with all dangerous permissions in the app and how they are being used. For example, dangerous permissions may be directly used by the app in a function, or the app may merely check to see if the app already has access to the permission. M-Perm will provide the context of how the app is using this permission. Knowing the context of permission requests can be useful to a variety of researchers including

[1]https://code.google.com/p/dex2jar/
[2]http://ibotpeaches.github.io/Apktool/
[3]https://bitbucket.org/mstrobel/procyon/

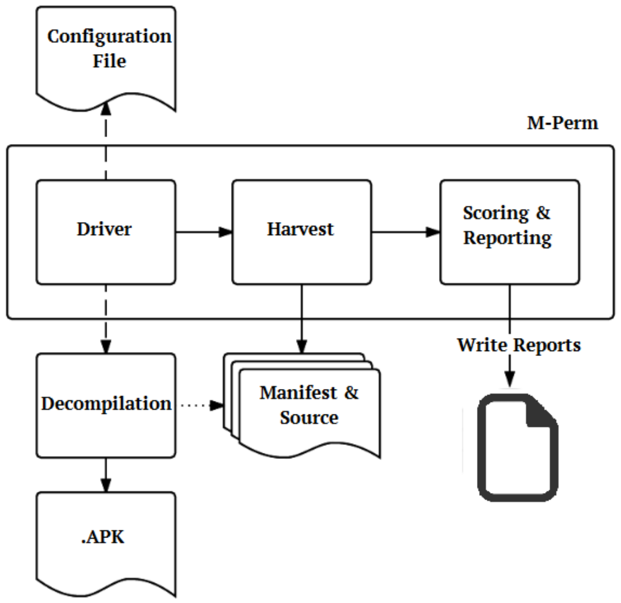


Figure 1: M-Perm Architecture

those analyzing app permission leaks [26], or those simply looking to better understand app permissions and how they are used.

The second file `source_report_<package_name>`, goes into more detail describing the usage of permissions inside of the app. The report lists the files that permission requests occur in, along with the block of code that is requesting the permission. For example, a permission snippet in this file could include an error or log message, checking to see if a permission has been granted, or even built in checks for possible malicious app usage. This output may be useful for both developers and researchers who would like to learn more about permission usage in either individual apps, or may be performed on a large set of apps to create a larger, aggregate data set.

Further documentation on the rest of these output files, along with other useful M-Perm information may be found on the project website.

4. EVALUATION

In order to assess the effectiveness and efficiency of M-Perm, we first evaluated it against a small set of oracle apps that we created specifically for this study. These apps contained pre-defined over and under permissions. We chose to evaluate M-Perm on several criteria including effectiveness (Precision, Accuracy, F-score & Accuracy), adaptability, and analysis time. In several cases, we compared M-Perm against Stowaway [15], a well-known Android Permission analysis tool. Although Stowaway was once likely regarded as the most effective tool for detecting permission gaps in Android apps, it is unfortunately several Android versions out of date, and according to its own website will likely produce inaccurate results. However, we still chose to display the results of M-Perm against Stowaway since it is likely the most well-known permission analysis tool with over 800 citations, and that it is a freely available open source tool.

Effectiveness: There are several ways to evaluate a program's effectiveness, including measuring the precision, recall, accuracy and F-score of the tool. Before the completion of M-Perm, we created an oracle set of apps to test the effectiveness of the permission analysis portion of the tool. This included 5 APKs that referenced the permissions that were requested in the manifest, and 5 APKs that did not reference the permissions that were requested in the manifest. The permissions we tested for included `READ_CALENDAR`, `ACCESS_FINE_LOCATION`, `CAMERA`, `READ_CONTACTS`, and `SEND_SMS`; all permissions which are considered to be 'dangerous', and must be requested directly by the app for them to be used. We chose to create an oracle since we could more easily and accurately control permission usage in sample apps which we created. Since M-Perm only works on Marshmallow 6.0 or above apps, all created apps were targeted for API 23. We then ran both Stowaway and M-Perm on this oracle of apps and evaluated their effectiveness in properly identifying over and under permissions. These results are shown in Figure 2.

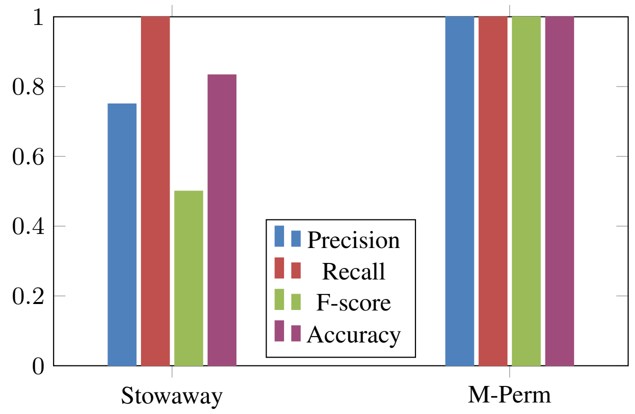


Figure 2: Tool Comparison on Oracle Apps

These results demonstrate the effectiveness of M-Perm to discover misused permissions in a small set of 10 custom apps in terms of precision, recall, accuracy and F-score.

Our next goal was to evaluate M-Perm on a larger set of exsiting Android apps. We began by collecting 50 apps targeted for Android Marshmallow from both Google Play[4] and apkMirror[5]. A few of these apps included Uber, Skype, LinkedIn, Twitter, and Google Calendar. We next examined each of these apps using M-Perm with the goal of ensuring that the tool was able to collect a variety of permissions related information about each app. The results of this analysis are shown in Table 1.

Table 1: Analysis Results for 50 Apps

	Total	**avg/App**
Total Permissions	1119	22.4
Dangerous Permissions	531	10.6
Third Party Permissions	321	6.4
Under Privileges	89	1.8
Over Privileges	110	2.2

Previous research has found that roughly a third of all apps contain at least one overprivilege [15]. In our very limited analysis, we found that less than 20% of collected apps contained an overprivilege, which indicates that the overprivilege rate is dropping in Android M apps using the new permission structure. However, our sample size is very small, and calls for future work in this area.

[4]https://play.google.com/
[5]https://www.apkmirror.com

Although far from a comprehensive study, our brief analysis demonstrates the capabilities of M-Perm along with serving as an indicator that the permission gap is still an issue in the reorganized permission model.

Adaptability: Future changes to the Android OS and its permission structure are inevitable. Therefore, it is imperative to create tools which can easily grow and adapt with Android. An important goal of our tool was to ensure that it was effective not only for the current version of Android M (6.0), but would also an effective tool with future versions of Android as well. M-Perm relies upon an app configuration file in order to identify *Normal* and *Dangerous* permissions, along with the groups which they belong to. This configuration file should make M-Perm easily adaptable for unforeseen permissions and grouping changes in future offerings of the Android OS.

Speed: Whether analyzing one app, or one thousand apps, the speed of an analysis tool is important. The vast majority of analysis time required by M-Perm was due to the reverse engineering process. For example, the Uber app took 288 seconds to be disassembled from an apk to .java source code, but took only about one second for M-Perm to analyze and create the final reports. We recorded the time required to complete the entire M-Perm analysis for each of the 50 real-world Android apps. On average, it took 328 seconds to complete the entire analysis process, with the shortest app taking 87 seconds to complete, and the longest taking 775 seconds. We expect that future improvements in the reverse engineering process will help to make the tool faster. However, it should be noted that numerous Android static analysis tools take much longer to complete. For example, components of PScout [9] are noted to take approximately half a day to complete [5].

5. TOOL AVAILABILITY

M-Perm is publicly available on our project website: **http://www.m-perm.com**. Our site contains the application, information on how to use the tool, and a link to the source code which is available on our GitHub page. We have also created a step-by-step guide for using our tool. Although we believe our tool has an easy to follow installation process, we have also made our tool available for download in a Ubuntu 16.04 LTS Virtual Machine (VM) which is available on our project website. We encourage first time users of M-Perm to use the tool through this provided VM.

6. RELATED WORK

The dangers of the Android permissions gap has received considerable attention over the last few years. In 2011 Felt *et al.* [15] surveyed 940 apps from the Android market and found that about one-third of apps contained at least one overprivilege. Wu *et al.* [28] found that vendor customizations on Android devices contain a significant amount of security vulnerabilities, largely due to 85% of all pre-loaded apps in stock images containing overprivileges. Wei *et al.* [27] analyzed third party and pre-installed Android apps and found that a high number of apps violated the principle of least privilege and that apps tend to add more privileges with each released version.

Felt *et al.* [15] conducted research to determine the cause of the permission gap and found that Android developers often mistakenly add unnecessary privileges in a counterproductive and futile attempt to make the app work properly, or due to confusion over the permission name they add it incorrectly believing its functionality is necessary for their app.

Although M-Perm is the first permission analysis tool directly targeted towards the new Android permission model, there are several tools which have been developed to assist in the decision making, permission process for developers. Stowaway is a powerful static analysis tool which has been used in previous research [17, 23,25], but it does suffer from drawbacks. Stowaway's own authors state that the tool only achieves 85% code coverage [15]. Additionally, the tool does not appear to have been modified to support more recent versions of the Android API.

Stowaway and PScout [9] both analyze the Android permission systems and provide a permission-API mapping using static analysis. Pscout has been used in several areas including the creation of a list of system calls requiring permissions [11, 16], research on privilege escalation attacks [9], and assisting with app risk assessments [22]. Unlike M-Perm, PScout is not intended to run on specific Android apps. The goal of PScout is to extract the mappings between the permissions defined in the Android operating system, and its API calls defined in primary jar files. Permlyzer is a tool which was built to determine where permissions are utilized in Android applications by using a mixture of static and runtime analysis [29].

7. LIMITATIONS & FUTURE WORK

One of the most significant limitations of M-Perm is that it only works with Android M 6.0 apps and higher. Although this represents the future direction of Android, many apps have yet to be converted to use the new model and only 10% of current Android devices are running Marshmallow [1]. This low conversation rate is not surprising due to Android's fragmentation problem [8].

M-Perm is largely dependent on 3rd party tools for reverse engineering target apps. These external tools frequently suffer issues which prevent extracting the necessary functionality from these apps, which limits the effectiveness of M-Perm. Future improvements may be made to increase both the speed, and quality of the reverse engineering process. It is also not fully understood how M-Perm will work with all variations of obfuscated code, although no known issues have been encountered thus far.

A primary goal for the development of the tool was to apply it to a large scale study to analyze permission usage in Android apps. The tool was designed to be robust, easily extensible, and fast. In conjunction with the use of existing tools, future work may be conducted to analyze a large number of Android apps collected from various locations such as Google Play, the Amazon app store[6], F-Droid[7], and AppsAPK[8] to conduct such work as comparing the rates of over and under privileges between Pre-M and Marshmallow apps, which has likely changed due to the reorganization of this permission model

8. CONCLUSION

We have created an Android permission analysis tool, M-Perm, which is able to identify permission usage and permission gap issues in the new Android permission structure. We evaluated our tool against 10 oracle apps and found that it is able to both correctly identify permission gap issues in these apps, and how the permissions are used as well. We also evaluated our tool against 50 publicly available apps collected from the web. M-Perm is available on our project website: **http://www.m-perm.com**

[6]www.amazon.com/appstore

[7]https://f-droid.org/

[8]http://www.appsapk.com/

9. REFERENCES

[1] Android marshmallow finally hits 10% adoption. http://www.informationweek.com/mobile/android-marshmallow-finally-hits-10--adoption/a/d-id/1325836.

[2] Number of apps available in leading app stores as of november 2015. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[3] <permission> android developers. https://developer.android.com/guide/topics/manifest/permission-element.html.

[4] <permission> android developers. https://developer.android.com/guide/topics/manifest/manifest-intro.html.

[5] Pscout github. https://github.com/zd2100/PScout.

[6] Requesting permissions at run time. https://developer.android.com/training/permissions/requesting.html.

[7] System permissions. http://developer.android.com/guide/topics/security/permissions.html.

[8] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith. Sok: Lessons learned from android security research for appified software platforms. 2016.

[9] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: Analyzing the android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 217–228, New York, NY, USA, 2012. ACM.

[10] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus. Automatically securing permission-based software by reducing the attack surface: An application to android. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 274–277, New York, NY, USA, 2012. ACM.

[11] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal analysis of android ad library permissions. *arXiv preprint arXiv:1303.0857*, 2013.

[12] M. Conti, V. T. N. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for android. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 331–345, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 346–360, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] J. Edwards. iphone lost market share to android in every major market except one. http://www.businessinsider.com/apple-ios-v-android-market-share-2016-1?r=UK&IR=T, January 2016.

[15] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[16] S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F. Shen, S. Y. Ko, and L. Ziarek. Flow permissions for android. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 652–657, Nov 2013.

[17] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein. Dr. android and mr. hide: Fine-grained security policies on unmodified android. 2011.

[18] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou. Analysis of android applications' permissions. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*, pages 45–46. IEEE, 2012.

[19] S.-H. Lee and S.-H. Jin. Warning system for detecting malicious applications on android system. In *International Journal of Computer and Communication Engineering*, 2013.

[20] Y. Ma and M. S. Sharbaf. Investigation of static and dynamic android anti-virus strategies. In *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*, pages 398–403, April 2013.

[21] M. Nauman, S. Khan, and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[22] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security*, volume 13, 2013.

[23] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 71–72, New York, NY, USA, 2012. ACM.

[24] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[25] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries.

[26] D. Wang, H. Yao, Y. Li, H. Jin, D. Zou, and R. H. Deng. Cicc: a fine-grained, semantic-aware, and transparent approach to preventing permission leaks for android permission managers. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, page 6. ACM, 2015.

[27] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM.

[28] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang. The impact of vendor customizations on android security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, pages 623–634, New York, NY, USA, 2013. ACM.

[29] W. Xu, F. Zhang, and S. Zhu. Permlyzer: Analyzing permission usage in android applications. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 400–410, 2013.

[30] S. Yerima, S. Sezer, and G. McWilliams. Analysis of bayesian classification-based approaches for android malware detection. *Information Security, IET*, 8(1):25–36, Jan 2014.

APPENDIX

A. WALK THROUGH

We will provide examples of how to both install the application from the project's source, along with using the provided Virtual Machine, which is the suggested method of operation.

A.1 Virtual Machine

Using the provided Virtual Machine (VM) is recommended for first time users of M-Perm. The following are instructions for using the provided VM.

1. **Download the VM Player:** The Ubuntu 16.04 LTS VM is built as a *.vmwarevm* file which may easily be opened using VMPlayer[9].
2. **Download the VM:** The Ubuntu 16.04 LTS VM is available on our project website: **http://www.m-perm.com**
3. **Choose Target App to Analyze:** M-Perm is capable of performing a permissions analysis on Android M 6.0 (API 23) apps. Although we have provided many popular APK files in the VM for analysis, the user may choose to examine any Android M app that they choose. In or VM, we use the `Desktop/mperm/MPermission/sample_apks` directory. The user should be cautious to only analyze Marshmallow apps using M-Perm, as older apps may lead to unexpected results.
4. **Reverse Engineer Target App:** The next step is to extract the necessary files from the APK file or analysis. This is the most time consuming process of M-Perm. The following command should be used to begin the reverse engineering process:

   ```
   python3 mperm.py d [-decompile]
   apk_path
   ```

   Figure 3 shows the execution of the reverse engineering portion of the app.

5. **Run Analysis Portion:** The analysis portion of M-Perm will only take a few seconds to run, and may be ran using the following command:

   ```
   python3 mperm.py a [-analyze]
   decompiled_apk_path
   ```

   Figure 4 demonstrates the analysis portion of M-Perm.

6. **Analyze Output:** The previous analysis portion of M-Perm produces the output of two .txt files:
   (a) `analysis_<package_name>`
   (b) `source_report_<package_name>`

   These are located in the *reports* directory of M-Perm. A sample output of Facebook Lite `analysis_<package_name>` is shown in Listing 1.

[9]https://www.vmware.com/products/fusion/fusion-evaluation

Listing 1: Example M-Perm Output

```
——————————— Analysis Report ———
Package: com.facebook.lite

——————— Permissions from Manifest ———
  0 android.permission.READ_CONTACTS
  1 android.permission.CALL_PHONE
  2 com.facebook.receiver.permission.ACCESS
  3 android.permission.WRITE_CALENDAR
  4 android.permission.READ_CALENDAR
  5 android.permission.CHANGE_WIFI_STATE
  6 android.permission.CHANGE_NETWORK_STATE
  7 com.facebook.orca.provider.ACCESS
  8 android.permission.ACCESS_WIFI_STATE
  9 android.permission.INTERNET
 10 com.facebook.lite.permission.C2D_MESSAGE
 .................
```

A.2 Installation from Source

Although using the provided Virtual Machine is recommend, users may install M-Perm from the publicly available source code. M-Perm was written in Python in order to make it as platform independent as possible. Even though we are providing instructions on installing the tool, the user should be cautioned that depending on their platform and configuration, they may be required to install components which are not described in this instruction set. We have tested M-Perm in Ubuntu 16.04 LTS and OSX 10.10 using Python 3.4.3.

1. **Get Source Code:** The application may be cloned from this Git Repository: https://github.com/dan7800/MPermission.git
2. **Install necessary submodules:** M-Perm uses pre-written decompilation scripts from the kocsenc/android-scraper project. After cloning the project, it can be installed using the following commands:

   ```
   git submodule init
   ```

   ```
   git submodule update
   ```

3. **Dependency Set Up:** After cloning and updating the submodule, run:

   ```
   cd android-scraper/tools/apk-decompiler;
   python3 setupDependencies.py
   ```

   Then navigate back to the primary M-Perm directory.

4. **Install Dependencies:** M-Perm's dependencies may be installed using the following command:

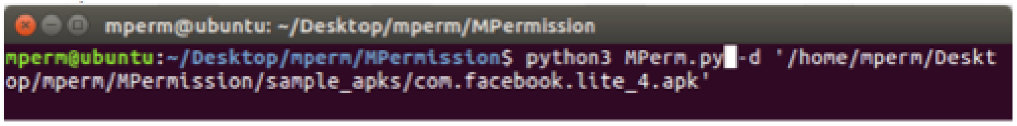
   ```
    pip install -r requirements.txt
   ```

   Note: Depending on your environment and what you have installed, the `requirements.txt` dependency list may need to be altered.

5. **Run the application:** You should now be able to run M-Perm using the following commands:

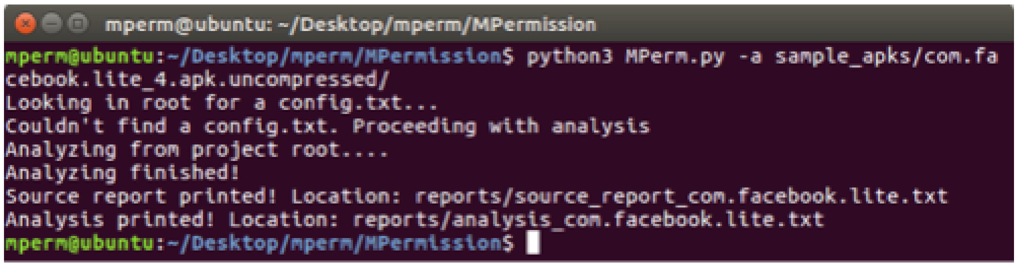
   ```
   python3 mperm.py d [-decompile]
   apk_path
   ```

   ```
   python3 mperm.py a [-analyze]
   decompiled_apk_path
   ```



```
mperm@ubuntu: ~/Desktop/mperm/MPermission
mperm@ubuntu:~/Desktop/mperm/MPermission$ python3 MPerm.py -d '/home/mperm/Deskt
op/mperm/MPermission/sample_apks/com.facebook.lite_4.apk'
```

Figure 3: M-Perm Reverse Engineering Process



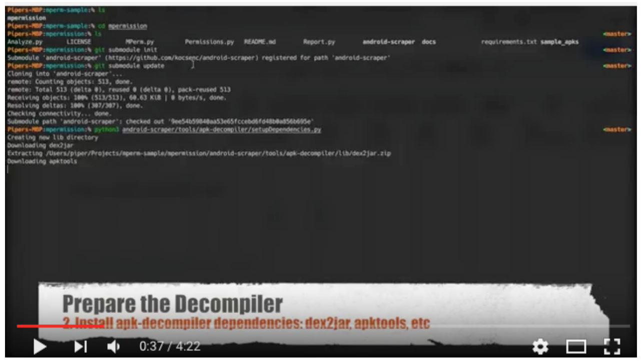
```
mperm@ubuntu: ~/Desktop/mperm/MPermission
mperm@ubuntu:~/Desktop/mperm/MPermission$ python3 MPerm.py -a sample_apks/com.fa
cebook.lite_4.apk.uncompressed/
Looking in root for a config.txt...
Couldn't find a config.txt. Proceeding with analysis
Analyzing from project root....
Analyzing finished!
Source report printed! Location: reports/source_report_com.facebook.lite.txt
Analysis printed! Location: reports/analysis_com.facebook.lite.txt
mperm@ubuntu:~/Desktop/mperm/MPermission$ 
```

Figure 4: Running M-Perm Analysis

Once completed, follow the instructions to the *Reports* location for the tool's two output files: `analysis_<package_name>` and `source_report_<package_name>` which are located in the *reports* directory of M-Perm.

B. SCREEN-CAST

A link to a screencast of the tool may be found at:
https://youtu.be/Wz6L2D3dFKs



```
Pipers-MBP:mperm-sample:% ls
mpermission
Pipers-MBP:mperm-sample:% cd mpermission
Pipers-MBP:mpermission:% ls                                                                    <master>
Analyze.py      LICENSE       MPerm.py       Permissions.py  README.md      Report.py      android-scraper  docs          requirements.txt sample_apks
Pipers-MBP:mpermission:% git submodule init                                                    <master>
Submodule 'android-scraper' (https://github.com/kocsenc/android-scraper) registered for path 'android-scraper'
Pipers-MBP:mpermission:% git submodule update                                                  <master>
Cloning into 'android-scraper'...
remote: Counting objects: 513, done.
remote: Total 513 (delta 0), reused 0 (delta 0), pack-reused 513
Receiving objects: 100% (513/513), 60.63 KiB | 0 bytes/s, done.
Resolving deltas: 100% (307/307), done.
Checking connectivity... done.
Submodule path 'android-scraper': checked out '9ee54b59840aa53e65fccebd6fd48b0a856b695e'
Pipers-MBP:mpermission:% python3 android-scraper/tools/apk-decompiler/setupDependencies.py     <master>
Creating new lib directory
Downloading dex2jar
Extracting /Users/piper/Projects/mperm-sample/mpermission/android-scraper/tools/apk-decompiler/lib/dex2jar.zip
Downloading apktools
```

Prepare the Decompiler
2. Install apk-decompiler dependencies: dex2jar, apktools, etc
0:37 / 4:22

Figure 5: Youtube Screencast