PERMITME: Integrating Android Permissioning Support in the IDE

Emmanuel Bello-Ogunu and Mohammed Shehab
University of North Carolina at Charlotte
{*ebelloog,mshehab*}@uncc.edu

Abstract

One of the most common security & privacy issues concerning mobile applications is the unnecessary access to sensitive information and resources. In a mobile application platform like Android, where a permission mechanism is used to maintain access control, the app developer dictates what permissions are necessary at install time. For various reasons however, including user confusion and lack of proper documentation, developers may overcompensate for the necessary permission. By this we mean developers often incorporate more permissions than are necessary for an app to function, thus undermining the access control mechanism and increasing the potential risk from a vulnerability exploit where sensitive user information is compromised. Even when developers intentionally include extra permissions, we believe it still the duty of a developer to at least be aware of what is at stake when it comes to collecting user information. In this paper we present PERMITME, a tool developed as a plugin for the Eclipse IDE, to interactively guide developers on the set of required permissions when creating Android applications. We conducted a between-groups user study in order to evaluate the effectiveness, efficiency, and usability of the PERMITME tool in enhancing the developer's experience when deciding to include Android permissions in their mobile applications.

Categories and Subject Descriptors D.4.6 [*Security and Protection*]: Access Controls; K.6.5 [*Security and Protection*]: Authentication

Keywords Android, permissions, software development, least privilege

ETX '14, October 21, 2014, Portland, OR, USA..
Copyright © 2014 ACM 978-1-4503-2530-1/14/10. . . $15.00.
http://dx.doi.org/10.1145/2688130.2688135

1. Introduction

Android provides a number of security features in its OS to reduce the frequency and impact of security issues among the plethora of apps made available to the public [6]. For example, a key design point of the application framework is that all applications, both pre-installed and third-party, are executed in a sandbox, meaning they are run in isolation from other apps. This forces all apps to explicitly request to share resources and data, and the only way to do this is to abide by a "permission" mechanism, where they statically declare the permissions they require to access protected APIs on the device. These permissions range from reading or writing the user's private data and connecting to the internet, to keeping the device awake or causing it to vibrate on certain events. Furthermore, the protected APIs include Camera functions, Location data (GPS), Bluetooth functions, Telephony functions, SMS/MMS functions, and Network/data connections. The user is then prompted to consent to the requested permissions at install time. It is interesting to note that Android does not grant permissions dynamically–at runtime–because of the belief that this will interrupt the user experience, further jeopardizing overall security [6].

Though there is evidence that presenting permission information to the user in a clear, more context-dependent way can influence mobile phone users in choosing apps that request fewer permissions [10], ultimately users still tend to make poor privacy and security decisions [1]. Moreover, previous research in the general area of usable security states that the burden of preserving privacy should not be placed solely on users [5]. As a result, we believe that code developers should take some responsibility in safeguarding users' privacy and preventing data leakage. One way to do this is by enforcing the concept of "least privilege" [11] in application development. Specifically, this principle deals with providing the minimal amount of access to information and resources necessary to accomplish an objective. In this context, we are addressing the permission model in Android applications. Fewer permissions means a more effective permission system [12, 15], so developers should apply this concept to the permission model. We believe the support of a plugin within an Integrated Development Environment (IDE) like the “Eclipse” platform that coaches users in situ can aid in this endeavor. Moreover, tying the permission metadata to the actual application code can be especially critical when the app changes over time, since it might otherwise accumulate stale permissions.

Given that previous research in the specific domain of mobile application security has suggested this “least privilege” concept before [2, 7, 13] but has not provided a full empirical study with users to support these claims, we intend to conduct a series of user studies in order to determine whether the use of an Eclipse plug-in built to provide feedback on missing or extraneous Android permissions can help developers code their apps with a better concept of ”least privilege” in mind. In this preliminary study, we survey participants who develop Android apps in order to measure how useful and usable the plugin is in providing the right cues or guidance. The applications used for evaluation were small, to demonstrate that even with minimal interaction with the plugin, we can begin to instill privacy-preserving behavior.

2. Android Permission Model

Categorizing Permissions. Android permissions can be defined by two different categories: *Protection Level* and *Functionality Group* [15]. There are four main Protection Levels: `Normal`, `Dangerous`, `Signature`, and `signatureOrSystem`. `Normal` permissions present minimal risk to Android apps and are granted automatically without need of the user’s explicit approval. `Dangerous` permissions allow access to personal sensitive information and various device features. These permissions are the only ones displayed to the user, as they always require user consent. `Signature` permissions are those associated with apps that are signed by the device manufacturer’s certificate only; these comprise the highest permission level. Lastly, `signatureOrSystem` permissions are granted to apps that are in the Android system image or signed with the same certificate. It is important to note that third-party apps are only permitted to use Normal and Dangerous permissions, while pre-installed apps may use permissions from all four protection levels.

Using and Enforcing Permissions. In order to request the appropriate permissions to access protected APIs, you must declare them in the `Android Manifest` XML file of an application’s source code. Figure 1 is an example from the Android Developers documentation of how to make such a declaration [6]. These are optionally granted at install time, and can be enforced at a number of points during the app’s operation, including: (1) at the time that a corresponding API call is made, (2) when the app is starting an activity (such as using Intents), (3) when accessing and operating on a Content Provider, which shares data between applications, and (4) during both sending and receiving broadcasts.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    ........
</manifest>
```

Figure 1: Declaring a permission to monitor incoming SMS messages (from the Android Developers site)

3. Our Proposed Approach

We propose PERMITME, which is a tool built as a plugin for the Eclipse IDE for static analysis on Android applications. It is meant to enforce “least privilege” by providing feedback to developers on missing or extraneous Android permissions. Currently the Android ADT does not have any built-in support for permission management, but our plugin supports both Eclipse ADT and the standard Eclipse Java EE platform. It is comprised of an API-permission mapping database and a static analysis engine. The API-permission mapping database was composed by combining the mappings generated by both STOWAWAY [7] and PSCOUT [2]. In this way, our mapping contains the different mappings for each of the Android versions from 2.6 to 4.0. These mappings are stored in a predefined text file which is used to generate a HashMap at the start of the plugin, which is stored in a static variable so that it can be referenced throughout the project. The map is of type HashMap<String, APIBean>, where the String is an API and APIBean is a Bean having a list of all the permissions associated with that API.

An initial implementation of PERMITME relied on an Abstract Syntax Tree (AST) to scan Java source files and find API calls associated with the app. An AST is a detailed tree representation of the Java source code, which is provided by Eclipse as a part of its Java Development Tools (JDT) parser[14]. The AST generated actually defines API to modify, create, read and delete source code. Each Java source file is represented as a subclass of the ASTNode class, and each specific AST node provides additional information about the object it represents. However, we found that working with the generated AST did not provide the full signature of methods as we expected them, which would cause trouble when trying to identify the exact API methods being called in the source code. Due to this limitation, we moved away from using AST, and instead relied on another method to allow the static analysis engine to scan class files. Therefore the plugin analyzes the corresponding byte code using a Visitor design pattern implementation of the ASM Java manipulation library [4].

Since we need to capture each class instances and their underlying methods we use ASM to detect the API methods of the project. ASM provides two visitors: ClassVisitor and MethodVisitor which helps in identifying the API methods. The plugin locates the .class files and each one is read using a ClassVisitor, which identifies all the method instances present. For each method identified a MethodVisitor is invoked, which cross-checks it against the list of all the API methods stored in the predefined mapping. If any method matches from the list, the corresponding permission is stored. This stored permission is checked against the currently requested permissions in the app, which are extracted by parsing the `AndroidManifest.xml` file, to see which are used (*required*) or unused (*extraneous*). If it is not present, these are considered *missing*, and a marker is generated in Eclipse at that point. The method visitor also identifies the line number and the source of the method. While generating the marker, a quick fix is also provided at that line to resolve the marker. A resource change listener is attached to the plugin so that any changes to the markers or the manifest file are captured in real time and the corresponding markers are updated.

In addition to the markers, the plugin uses the console to present the developer with a summary of all the current markers in both the code and the `AndroidManifest.xml` file, as in Figure 2(e). This summary includes the currently existing, required and the redundant (extraneous) permissions. The developer can also use this summary to activate a Quick Fix. Any Missing permission-related error will prevent the code from being built, until the developer corrects the issue and reruns the plugin.

4. Methodology

We recruited 20 participants from a Mobile Application Development course being taught during the Fall 2013 semester. Of them, 60% reported an Advanced level or programming knowledge/expertise, 35% reported Moderate, and 5% reported Novice. We performed the study in an on-campus lab; participants were given debugging-related tasks for two Android applications, one requiring one permission, and the other requiring four, and both having extraneous ones. Participants were randomly assigned to one of the two conditions, either with or without use of the plugin. They were observed unobtrusively, and made aware at start of the survey that results were anonymous. Time to Complete, along with Number of Missing Permissions and Number of Extra Permissions, are the measures upon which the study results were evaluated. Additionally, the survey results provided an evaluation of the Usability of the PermitMe plugin, in the areas of Ease of Use, Trustworthiness of results, Effectiveness in accomplishing what was intended, and Readability of output. We hypothesize that the PermitMe plugin is more *efficient*, *effective*, and *usable* in helping a developer reduce the number of extraneous permissions and incorporate any missing permissions than other resources.

5. Results

5.1 User Study Results

Accomplishing the user study tasks consisted of finding the missing permissions and correcting the code in a way that allowed it to run successfully. For task 1, the mean completion time for participants who did not use the PermitMe plugin was three times higher than those who used the plugin. Since not all of our participant data was normally distributed, we used the Mann-Whitney-Wilcoxon Test to compare the two groups, which still confirmed this was a statistically significant difference (p-value <0.001). With Task 2, we found similar results, with a Wilcoxon test producing a p-value <0.001. Given that completion time is our measure of efficiency, these results support our hypothesis. Table 1 shows the mean time comparisons for tasks 1 and 2.

There was no significant difference between the Missing Permissions and Extra Permissions for Task 1. This was expected given that there was only one Missing permission and one Extraneous permission. For Task 2, the mean Number of Missing Permissions for non-plugin users was 2.7, and 0 for plugin users. Using a Wilcoxon test, we found statistically significant difference between the two groups (p<0.001). The mean Number of Extra Permissions for non-plugin users was 1.1, and 0.4 for plugin users. Using a Wilcoxon test, we found statistical significance between the two groups (p<0.05). This also supports the claim of effectiveness in our hypothesis.

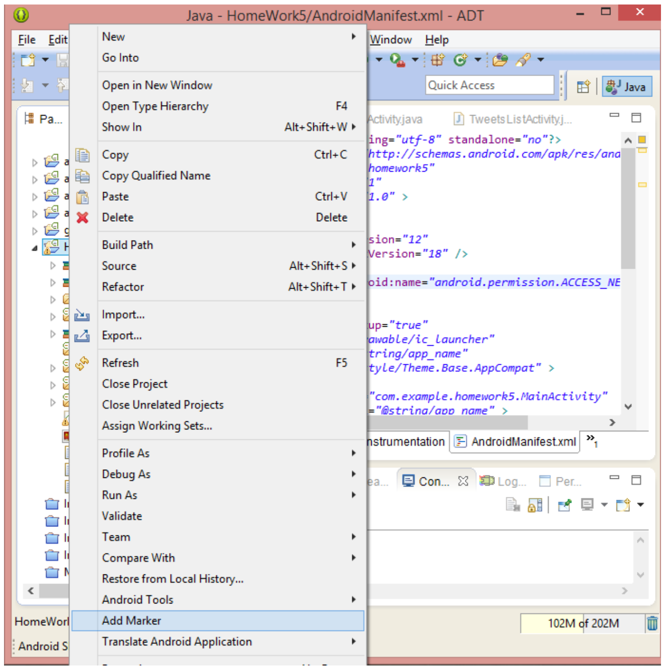
Table 1: Comparison Results for Tasks 1 and 2

Measure	**Without Plugin** (μ, σ)	**With Plugin** (μ, σ)	**p-value**
Task 1			
Time (seconds)	(722.5, 194.75)	(234.8, 35.14)	< 0.001
No. Missing Perms	(0.1, 0.32)	(0.0, 0.0)	0.3434
No. Extra Perms	(0.1, 0.32)	(0.1, 0.32)	1
Task 2			
Time (seconds)	(792.3, 239.51)	(406.6, 115.41)	< 0.001
No. Missing Perms	(2.7, 1.16)	(0.0, 0.0)	< 0.001
No. Extra Perms	(1.1, 0.88)	(0.4, 0.52)	0.046

5.2 Survey Results

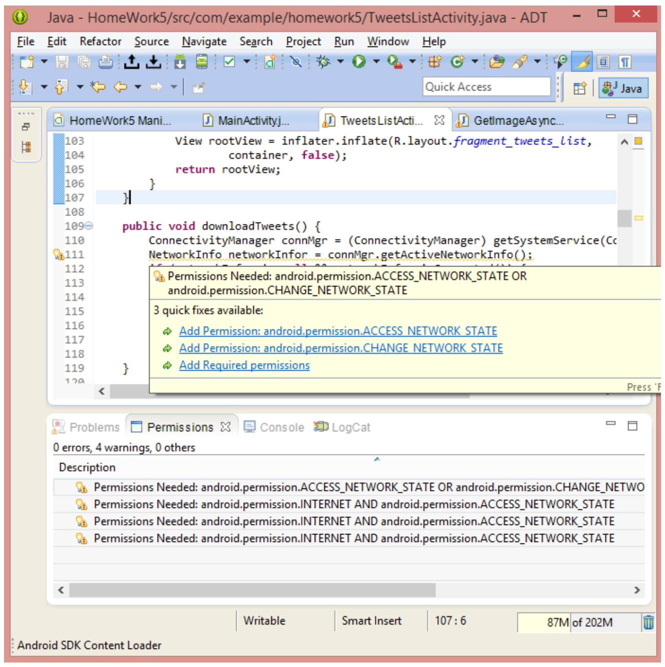
In this subsection we present the survey results that investigated our plugin's ease of use, trustworthiness, effectiveness, and readability. The exit survey included a number of 5-Point Likert-scale questions concerning the usability of our plugin, comparing it to online resources (ex. Stack Overflow), and the Android logging tool (LogCat). The results of the Wilcoxon tests performed on these question responses are presented in Table 2.

When comparing our plugin to online resources, there was no significant difference with regard to Trustworthiness, Effectiveness, and Readability. However, there was a significant difference (p-value <0.05) for Ease of Use, indicating that our plugin was easier to use when compared to online



```
ing="utf-8" standalone="no"?>
http://schemas.android.com/apk/res/ana
homework5"
1"
1.0" >

sion="12"
Version="18" />

oid:name="android.permission.ACCESS_NE

up="true"
awable/ic_launcher"
tring/app_name"
tyle/Theme.Base.AppCompat" >

"com.example.homework5.MainActivity"
="@string/app_name" >
```

(a) Running the Plugin



```
            View rootView = inflater.inflate(R.layout.fragment_tweets_list,
                    container, false);
            return rootView;
        }
    }

    public void downloadTweets() {
        ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Co
        NetworkInfo networkInfor = connMgr.getActiveNetworkInfo();
```

Permissions Needed: android.permission.ACCESS_NETWORK_STATE OR android.permission.CHANGE_NETWORK_STATE

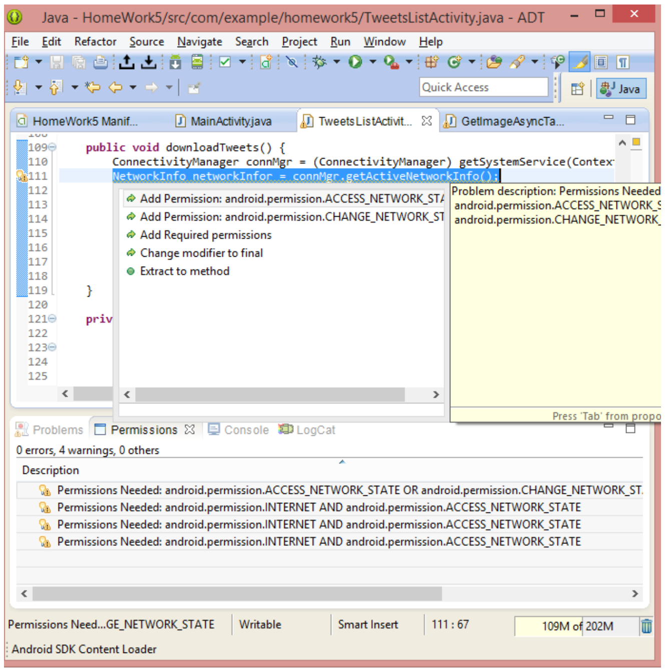
3 quick fixes available:

- Add Permission: android.permission.ACCESS_NETWORK_STATE
- Add Permission: android.permission.CHANGE_NETWORK_STATE
- Add Required permissions

0 errors, 4 warnings, 0 others

Description
Permissions Needed: android.permission.ACCESS_NETWORK_STATE OR android.permission.CHANGE_NETWO
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE

(b) Marker Indicating Permission Related API Calls



```
    public void downloadTweets() {
        ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Contex
        NetworkInfo networkInfor = connMgr.getActiveNetworkInfo();

    }

    priv
```

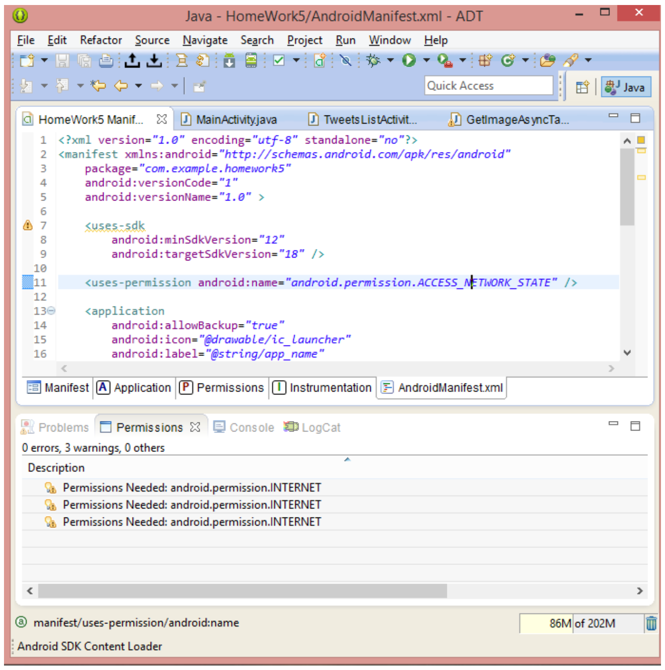
- Add Permission: android.permission.ACCESS_NETWORK_ST
- Add Permission: android.permission.CHANGE_NETWORK_ST
- Add Required permissions
- Change modifier to final
- Extract to method

Problem description: Permissions Needed android.permission.ACCESS_NETWORK_S android.permission.CHANGE_NETWORK

0 errors, 4 warnings, 0 others

Description
Permissions Needed: android.permission.ACCESS_NETWORK_STATE OR android.permission.CHANGE_NETWORK_ST.
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE

(c) Right-click to Access Quick Fix

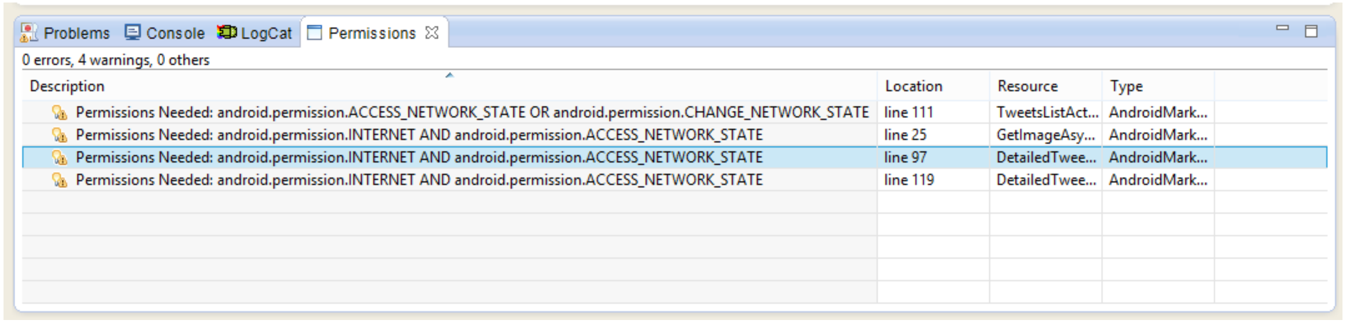


```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.homework5"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="12"
        android:targetSdkVersion="18" />

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
```

0 errors, 3 warnings, 0 others

Description
Permissions Needed: android.permission.INTERNET
Permissions Needed: android.permission.INTERNET
Permissions Needed: android.permission.INTERNET

(d) Permission Added to Android Manifest Addition



0 errors, 4 warnings, 0 others

Description	Location	Resource	Type
Permissions Needed: android.permission.ACCESS_NETWORK_STATE OR android.permission.CHANGE_NETWORK_STATE	line 111	TweetsListAct...	AndroidMark...
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE	line 25	GetImageAsy...	AndroidMark...
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE	line 97	DetailedTwee...	AndroidMark...
Permissions Needed: android.permission.INTERNET AND android.permission.ACCESS_NETWORK_STATE	line 119	DetailedTwee...	AndroidMark...

(e) List of Permission-related issues in code

Figure 2: Plugin Screenshots

resources. This can be attributed to the difficulty and complications involved in filtering through solutions from online resources such as Stack Overflow. These results are satisfactory, given that the average scores were relatively high, demonstrating that the plugin is as good as available online resources with regards to those three measures, and our plugin was significantly better as far as Ease of Use.

Table 2: Comparison with other Help Resources

Measure (5 Point Likert-Scale)	**Online Resources** (μ, σ)	**Our Plugin** (μ, σ)	**p-value**
Ease of Use	(3.86, 0.51)	(4.50, 0.71)	0.0239
Trustworthiness	(4.38, 0.72)	(4.30, 0.82)	0.8154
Effectiveness	(4.38, 0.81)	(4.60, 0.69)	0.4602
Readability	(4.50, 0.73)	(4.40, 0.84)	0.7607

Measure (5 Point Likert-Scale)	**LogCat** (μ, σ)	**Our Plugin** (μ, σ)	**p-value**
Ease of Use	(3.10, 1.29)	(4.50, 0.71)	0.0090
Trustworthiness	(4.10, 0.74)	(4.30, 0.82)	0.5744
Effectiveness	(3.60, 1.07)	(4.60, 0.67)	0.0258
Readability	(2.80, 1.14)	(4.40, 0.84)	0.0024

Considering the comparison between PERMITME and LogCat, the measures that proved significantly different in favor of the plugin were Ease of Use (p-value=0.009), Effectiveness (p-value=0.0258), and Readability (p-value=0.0024). This is an expected result as it is very difficult to read and use the Android LogCat output. Since Trustworthiness of LogCat and the plugin had means of 4.1 and 4.3 respectively, this meant the suggestions our plugin offered were as trusted by the users as that of the Android logging platform. Overall, the feedback provided by the plugin was received as well as or better than many of the common resources that developers rely for assistance, so this supports the usability claim of our hypothesis.

Overall, the permission related feedback provided by the plugin was received as well as or better than many of the common resources that developers rely for assistance, so this too supports our hypothesis, which stated the PERMITME plugin would be more *usable* than the other resources.

5.2.1 Qualitative Feedback

In addition to the quantitative responses obtained from the survey, there were snippets of textual feedback provided by users in the last question of the survey. This open-ended question asked for any additional comments about the plugin/user study. The responses were valuable not only in supporting our hypotheses, but also in providing potential solutions for improving the usability of the plugin. Below are a subset of the responses provided.

> *"I found the plugin to be very useful. Normally developer tends to add the required permissions in the manifest file but forget about the permissions that are extraneous. Making the use of this plugin [accessible] will allow users to get notified about the extraneous permissions."*

> *"Really good plug-in, I love it."*

> *"Logcat integration would help in the debugging process. I go there first when looking for a problem, but Logcat doesn't indicate it's a permissions problem causing the error, just a 'null pointer' exception."*

6. Related Work

IDE Support. A very popular tool developed a few years ago to detect bugs in software is known as FINDBUGS. This open-source program was developed by researchers from the University of Maryland. Unlike most formal methods, which focus on automating detection through narrow, sophisticated analyses, FINDBUGS focuses on using simple, broad techniques based on actual bugs in code, in order to better understand what kinds of bugs exist. [8]. Empirical results were presented on the effectiveness of FINDBUGS on real programs being used in production environments; this was based on an analysis on the number of findings that could be considered as *false positives*, *harmless bugs*, *dubious bugs*, or genuinely *serious bugs*. The target was that at least 50% of all reported bugs be genuine, and after many iterations [3, 9], positive results on the quality and significance of the code bugs were reported, but the research lacked an analysis of the usability of the tool, particularly considering developers of different expertise levels.

The research by Xie et al. [16] deals with a plugin for Eclipse that can detect and address common web application vulnerabilities, such as improper input validation, broken access control, and cross-site request forgery. This interactive approach is known as the Assured Software IDE (ASIDE) and it relies on two key techniques to aid programmers: *interactive code refactoring* and *interactive code annotation* [16, 17]. The idea here is that the aid developers receive should be provided as an in situ reminder, instead of after the program has been written. ASIDE achieves this by continuously monitoring activity and statically analyzing portions of the code at a time, in order to promptly provide feedback to code edits [16]. Like FINDBUGS, a critical component of determining the effectiveness of this plugin is the measure of its ability to find genuinely vulnerable code. Evaluation was conducted on commercial software through two comparison user studies between novice and professional developers [17], which highlighted some key issues in providing interactive support, including the need to more easily understand, address, and even dismiss provided warnings through the plugin.

Permissions & Privacy. Au et al. developed a version-independent permission plugin called PSCOUT–short for Permission Scout–that performs static analysis to retrieve the permission specification from Android applications [2]. Evaluation was done with a sample of 1,260 applications on the basis of completeness and soundness in determining overprivileged apps. It was compared against another research tool called STOWAWAY [7], and PSCOUT proved to be significantly more complete and sound in permission mappings, but ultimately there was no significant difference in overprivileging. PSCOUT is not without limitations however. It can't handle some API calls that are invoked through reflection (though admittedly neither can ours, as this is an open problem in Android development). The intent of this tool is different from ours, in that PSCOUT is simply a permission mapping plugin, whereas PERMITME is a plugin that uses a permission map to compare against API calls found in a developer's code, and assists them in adjusting their application's permissioning in order to require only what is necessary.

Vidas et al. also developed an Eclipse plugin called PERMISSION CHECK TOOL, that extracts the Android permission specification from an app and aids developers in utilizing least privilege during permissioning [13]. The functionality of the PERMISSION CHECK TOOL is similar to that of PERMITME. However, a major difference lies at the implementation level. Their API-to-permission mapping is built by parsing the Android API documentation, which is known to be incomplete. Furthermore, the plugin strictly analyzes source code, and there is no ready library or dataset of Android application source code readily available, so they did not present extensive empirical analysis of the plugin. Lastly, its permission map only covers Android 2.2. The PERMITME plugin described in the previous section is an improvement over the PERMISSION CHECK TOOL, with empirical data collected to support this claim.

7. Conclusion

In this paper, we presented PERMITME, a tool developed for integration within the Eclipse IDE for the purpose of developing more privacy-sensitive Android applications. This is accomplished by enforcing the principle of "least privilege" through minimization of permissions requested by an app. This work is being done in response to the need for a reduction in over-privileged apps, which comes largely from the need of developers to take on the responsibility of preserving user privacy, and the lack of clarity and completeness in Android and third-party API documentation when it comes to development. Our research does more than highlight this void, but makes an attempt to fill it through the use of our tool, and evaluate our solution's effectiveness with real developer input. We hypothesized that the PERMITME plugin would prove more efficient, effective, and usable than other resources. An empirical analysis based on the use of the plugin by 20 Android developers suggests that it was successful in this endeavor.

Acknowledgments

We thank Swapnil Thorat and Yogeshwar Reddy Anugu for their assistance in developing the PERMITME plugin. Mohamed Shehab's research was partially supported by grants from the National Science Foundation (NSF-CNS-0831360, NSF- CNS-1117411) and a Google Research Award. Emmanuel Bello-Ogunu was supported in part under the Department of Education (DoEd) Graduate Assistance in Areas of National Need (GAANN) Computing Scholars Fellowship as well as the NSF Scholarship for Service. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the DoEd or NSF.

References

[1] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *Security & Privacy, IEEE*, 3(1):26–33, 2005.

[2] 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*, pages 217–228. ACM, 2012.

[3] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou. Evaluating static analysis defect warnings on production software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 1–8. ACM, 2007.

[4] O. Consortium. ASM: a bytecode engineering library. `http://asm.ow2.org/index.html`, 12 Oct 2013.

[5] L. F. Cranor. *Security and usability: Designing secure systems that people can use*. O'reilly, 2007.

[6] A. Developers. Android developers: Permissions. `http://developer.android.com/guide/topics/security/permissions.html`.

[7] 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*, pages 627–638. ACM, 2011.

[8] D. Hovemeyer and W. Pugh. Finding bugs is easy. *ACM Sigplan Notices*, 39(12):92–106, 2004.

[9] D. Hovemeyer and W. Pugh. Finding more null pointer bugs, but not too many. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 9–14. ACM, 2007.

[10] N. Sadeh, L. F. Cranor, and P. G. Kelley. Privacy as part of the app decision-making process, 2013.

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

[12] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen. Asking for (and about) permissions used by android apps. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 31–40. IEEE Press, 2013.

[13] T. Vidas, N. Christin, and L. Cranor. Curbing android permission creep. In *Proceedings of the Web*, volume 2, 2011.

[14] L. Vogel. Eclipse JDT - Abstract Syntax Tree (AST) and the Java Model. `http://www.vogella.com/tutorials/EclipseJDT/article.html`, 08 Aug 2012.

[15] 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*, pages 31–40. ACM, 2012.

[16] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton. Aside: Ide support for web application security. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 267–276. ACM, 2011.

[17] J. Zhu, H. R. Lipford, and B. Chu. Interactive support for secure programming education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 687–692. ACM, 2013.