Eurographics Conference on Visualization (EuroVis) 2014
H. Carr, P. Rheingans, and H. Schumann
(Guest Editors)

Volume 33 (2014), Number 3

Papilio: Visualizing Android Application Permissions

M. Hosseinkhani Loorak[1] P. W.L. Fong[1] and S. Carpendale[1]

[1]Department of Computer Science, University of Calgary, Calgary, Alberta, Canada

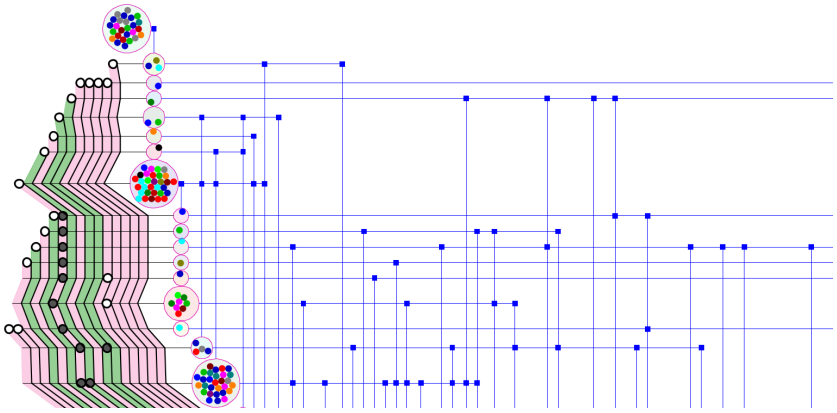

Figure 1: *The visualization of Android application permissions with Papilio*

Abstract

We introduce Papilio, a new visualization technique for visualizing permissions of real-world Android applications. We explore the development of layouts that exploit the directed acyclic nature of Android application permission data to develop a new explicit layout technique that incorporates aspects of set membership, node-link diagrams and matrix layouts. By grouping applications based on sets of requested permissions, a structure can be formed with partially ordered relations. The Papilio layout shows sets of applications centrally, the relations among applications on one side and application permissions, as the reason behind the existence of the partial order, on the other side. Using Papilio to explore a set of Android applications as a case study has led to new security findings regarding permission usage by Android applications.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Systems]: Information Interfaces and Presentation—User Interfaces

1 Introduction

We present Papilio as a new visualization for the permissions data of mobile applications. In particular, we examine data about the permissions used in Android applications. This research is motivated by the proliferation of mobile devices in the last few years, where third-party applications have seen ubiquitous adoption. There were at least 5,000 new applications added to the Android Market each month in 2013 [App12]. These applications empower people in many aspects of their lives such as social networking, online banking and self-monitoring. However, they may also put people at risk for invasion of privacy or even possibly malicious control of personal devices. We apply visualization to better understand these trade-offs.

While mobile applications bring many types of new conveniences into our everyday lives, most people are unaware if their privacy is being violated or whether an application has access to their private data. Examples of these types of

private data include contact lists, SMS messages and accurate geolocations. Recent reports show the existence of malicious applications in the official Android Market and Apple App Store [McA12, Sym13, Pan13]. In order to provide better protection from malicious applications, modern mobile platforms have utilized different approaches to minimize the risks. Apple, for instance, makes use of a *vetting process* [App13]. In this process, a trusted party ensures that each application is in accordance with the Apple's developer license agreement. This agreement contains the accepted conditions under which an application can get access to private data.

By contrast, Google does not verify Android applications. Instead, people are responsible for granting access to their sensitive data to Android applications by means of install-time permissions. Each application must declare its required permissions before its distribution. When a person initiates the process of installing an application, a list consisting of all the permissions requested by the application will be shown. This list alerts the individual to all resources to which the application will gain access if it is installed on the device. For instance, installing an application that requests SEND_SMS permission, allows the application to send SMS messages on behalf of the person who owns the device. Malicious applications might use this permission to send SMS messages to premium numbers. If one is not willing to grant all the requested permissions to the application, she can cancel installing the application. More details on how the permissions work in the Android platform will be provided in §3.1.

Although install-time permissions do provide people with control over their privacy and security, they can be ineffective if the developers do not follow the principle of least privilege in their permission requests [SS75]. Felt *et al.* [FCH*11] examined a dataset of Android applications and found that almost one-third of applications in their dataset are over-privileged, i.e. they requested more permissions than they need. The violation of least privilege and other potential issues in requesting permissions by Android applications can be discovered by performing exploratory analysis tasks. Information visualization (InfoVis) systems have been noted as potentially useful when applied for exploratory tasks in large information spaces [FVWSN08]. Thus, we present a new visualization technique, Papilio, for exploring how permissions are being requested by Android applications in practice.

Our main contribution is Papilio, which is a two-sided visualization that centrally presents Android applications grouped into equivalence classes (e-classes for brevity) according to their requested permission set. The superset and subset relations among e-classes, based on their set of requested permissions, are partially ordered. Papilio visualizes these partial order relations, henceforth called parent-child relations, on the right side of central e-classes. On the left side, the permissions, which are the reason behind the existence of the parent-child relation among e-classes, are shown as e-class attributes (Figure 1). Papilio also offers interaction options, which include Slide on Halo: a novel technique for spatial cognition of the off-screen points of interest and also fast and smooth navigation towards them.

The target audience of Papilio is individuals who aim to explore and analyze the permission usage of Android applications. Examples include security analysts, Android developers or even a mobile application user. In this paper, we consider security analysts as the main targeted operators of Papilio.

This paper is organized as follows. First, we present the related work and then provide a brief background on Android permissions in §3.1. We then describe our dataset of Android applications in §3.2 and the related security tasks in §3.3. We step through our iterative visualization development process in §4 and present a detailed description of the Papilio visualization technique and its interactions in §5. For demonstrating the utility of Papilio, we discuss security findings we obtained from exploring and analyzing Papilio in §6. Conclusions are drawn in §7.

2 Related Work

Two areas of research, information security and information visualization, both offer literature related to our work.

Information security: There is emerging research investigating the use of information visualization for security purposes [BKvOS10, BJL*13, TPP09].

In the security literature, we look at techniques that are closely related to our research in that they exploit visualization for security analysis of smart phone permissions. The first is by Balebako *et al.* [BJL*13] who propose visual interfaces to inform mobile users of any privacy leaks. In one of their proposed interfaces, a matrix representation of application names combined with the frequency of leaked information is depicted. The second work is by Barrera *et al.* [BKvOS10] who use Self Organizing Maps (SOMs) [KSH01], which incorporate a type of artificial neural network to visually observe permission usage in their targeted Android applications. An SOM can provide a low-dimensional visualization of the high dimensional data while preserving the topological properties of the input space.

The work above differs from ours in that Papilio provides a lossless presentation of the dataset. More specifically, every detail in our dataset is represented in our proposed layout. In contrast, techniques which involve a dimensionality reduction may hide important information. For instance, in Barrera *et al.*'s work, when the category of a neuron (a hexagonal region in the map) is labeled with only the strongest category (i.e. a winner-take-all approach) other weaker categories in that neuron are not visible for further analysis. In addition, the composite application permission usage vectors of each hexagon are not available in their map. Consequently, deeper analysis of each permission vector for every application would not be possible in a hexagon.

Information visualization: In our proposed visualization we have used an iterative design approach by applying different layouts in each iteration. These three different layouts

are generally inspired by different variations of node-link diagrams and adjacency matrices which are discussed in more detail in §4.

There is a large body of work in the literature regarding graph visualization [HMM00, VLKS*11]. However, investigations with our early design iterations revealed that our dataset has several densely connected e-classes in the form of a rooted Directed Acyclic Graph (DAG). DAGs are usually represented with Sugiyama layout [STT81], however, in large dense DAGs even an optimal Sugiyama layout can have many link crossings and long links [PNK11, BW11].

Concept lattices [Wil92] that are employed in Formal Concept Analysis (FCA) would be an example of utilizing a variant of Sugiyama layout to represent an information space in FCA. FCA is a well-known method for data analysis and knowledge representation. Eklund et al. [EV10] have done a comprehensive study of different types of concept lattices in FCAs. Within them, representation of many-valued contexts [MDNST08] would be the most relevant layout to our work. However, these types of layouts are usually considered for numerical or ordinal attributes of a concept, while in our work, permissions are nominal data attributes associated with applications. Furthermore, in many-valued contexts, increase in number of objects and their attributes will result in overcrowded layout with too small embedded visualizations for each concept [EV10]. A common solution to the problem in many-valued contexts, is having two-separated views of data, in which one view represents the binary relation between concepts while the other represents attributes of the selected concept in another separate layout. Lastly, in concept lattices, representations are usually based on a sole binary relation between concepts, while in our layout, three different binary relations between data entities, namely, application-permission relation, application-category relation, and permission-type relation are represented in a unified layout.

An alternative approach for graph representation is adjacency matrices. Ghoniem *et al.* [GFC04] note that while matrix representations do not have link crossing and node overlapping problems, following paths is relatively difficult.

Therefore, a hybrid utilization of node-link diagram and adjacency matrices could alleviate the weakness of each layout [HFM07]. Dinkla *et al.* [DWvW12] proposed compressed adjacency matrices (CAMs) for visualizing gene regulatory networks (GRNs). In their representation, the input graph and its nodes are decomposed and rearranged to some matrix-like blocks which are connected to each other with arcs. Compared to node-link diagrams, CAM layouts reduce link crossings. In addition, they are more space efficient in comparison with traditional adjacency matrices.

Another line of related work is GeneaQuilts [BDF*10], by Bezerianos *et al.* wherein they have applied Quilts layout [WBS*07] to genealogy data. In their representation of pedigrees, generations form a zigzagged sub-matrix of individuals. Families and their pedigree relationships are specified via paths and intersections of rows and columns within and across sub-matrices. Successive generations are linked to the previous generations by sharing family columns. The GeneaQuilts design also considers interaction techniques such as Bring & Slide, path highlighting, panning, and filtering.

Our work differs from theirs in the following respects. First, Papilio is a two-sided visualization technique in the sense that the parent-child relations between e-classes are represented on the right side, and the set of permissions for each e-class is shown on the left side as e-class attributes. This two-sided layout enables richer representation of information attached to each e-class. In contrast, the GeneaQuilts or CAM are layouts that do not need information richness for each element.

Second, our proposed visualization deals with a different type of dataset in its structure compared to Genealogy or GRN data. In our dataset, it is possible to have a skip link between two e-classes of applications belonging to two far non-successive layers of our representation. Therefore, the skip links are less localized. By contrast, in genealogy data, it is a rare case for an individual to live and make families for more than three or four generations. This distinct difference will be more noticeable when an individual wants to look at the off-screen children or parents of an e-class. We propose our Slide on Halo technique to address this issue in §4.

3 Android Permissions

In this section, we provide a brief background on Android permissions. We then present our dataset of Android applications and list the tasks in analyzing this dataset.

3.1 Background on Android Permissions

Android markets, such as Google Play [Goo13a] and Amazon Appstore for Android [Ama13], allow third-party developers to freely upload their developed application to the market without performing any security checks on them. Therefore, sensitive resources on a device might be endangered by potentially malicious applications. Contact list, photos and location information are some examples of sensitive resources that can affect an individual's privacy. However, to protect Android users, these resources are protected by ***permissions***. A permission is an abstract concept for binding operations to resources. For accessing and operating on each of the sensitive resources of an Android device, a permission is needed. For example, performing phone calls and taking pictures require PHONE_CALLS and CAMERA permissions, respectively. Generally permissions are either defined by the Android platform, henceforth called standard permissions, or by the application developers. The standard permissions protect system resources, while the permissions defined by a developer in an application restrict access to the application from other applications. In this work, we only consider standard permissions which are categorized into four protection levels [Goo13b].

Normal: A low risk permission that gives the application the ability to perform actions that do not have harmful con-

sequences for the Android user and her device. Examples include SET_WALLPAPER and VIBRATE.

Dangerous: A higher risk permission with potentially harmful effects on the user's sensitive resources. Examples include CAMERA, RECORD_AUDIO and READ_CONTACTS permissions giving the application the ability to use the device camera, record audio and read the user's contact list, respectively.

Signature: A permission that Android system assigns only to the applications signed with the same certificate as the application that declared the permission. As an example, the permission BRICK allows an application to disable the device. It is a signature level permission declared as a standard permission by Android. Therefore, only the applications signed by the Android device manufacturer should be able to obtain this permission.

SignatureOrSystem: A permission that can only be granted to applications in the Android system image or that are signed with the same certificate as the application that declared the permission. These restrictions should confine the use of signatureOrSystem permissions to the applications pre-installed on the Android device. Examples include INSTALL_PACKAGES which allows an application to install other application packages on the device.

For obtaining permissions, the developer declares the required permissions for her application in a file called Manifest.xml which is part of the Android application package. At install-time, the list of requested permissions by the application is presented to the user. At this point, the user must either grant all the permissions to the application and continue with the installation or give up installing the application.

3.2 Dataset

Since we could not obtain a publicly available dataset of Android applications which contains the category of each application, its standard requested permissions and the protection level of each permission, we gathered our own dataset.

In September 2012, we investigated the categories of the 200 most popular free applications in Google Play and ranked the categories based on their number of applications in descending order. We then selected the top 16 categories. Table 1 lists our chosen categories, and provides the number of applications per category in the set of 200 applications we explored. Afterwards, we downloaded the top ranked 25 free applications from each selected category. As a result, our dataset consists of 400 Android applications in total.

Each application in our dataset was collected in the form of an Android application package (APK). We used the Android Asset Packaging Tool (aapt) to extract the permissions requested by the application from its Manifest.xml file. We then filtered the standard permissions requested by the application. Finally, our dataset contains the name, category and the standard permissions of each application.

Our collected applications request 76 distinct permissions out of 130 available standard permissions. The remaining ones are never requested by any of our sample applications. Table 1 shows the average number of permissions requested by applications of each category within our dataset.

Table 1: *Top 16 categories of popular free applications in Google Play and average number of requested permissions by applications per category.*

App. Category	Num. of Apps. (in top 200 apps.)	Average Perms. (in our dataset)
Game	81	4.01
Communication	16	16.04
Social	13	8.80
Productivity	13	4.42
Entertainment	11	6.36
Tools	11	6.25
Music & Audio	10	7.24
Finance	8	3.87
Weather	5	4.73
Business	5	5.66
Travel & Local	5	5.86
Shopping	5	6.25
Health & Fitness	4	9.08
Lifestyle	4	5.01
Education	4	2.77
Medical	3	1.36

Furthermore, for each standard Android permission we extracted its protection level from AndroidManifest.xml file within the Android platform. Each permission is then presented with its name and protection level in our dataset.

3.3 Sample Tasks

When analyzing the permission usage of Android applications, an analyst might encounter a number of exploratory tasks. These tasks for data analysis vary depending on the goals each analyst has in mind. From working with security experts, we created a list of tasks that are important for analyzing our dataset. Our proposed visualization, Papilio, assists the analysts to perform these tasks and supports exploration of how Android applications request permissions in practice. The list of tasks is categorized as:

Explicit (X): tasks that their answers explicitly exist in the dataset.

X 1 : Identify the requested permission set of each Android application.
X 2 : Find the protection level of each requested permission by an application.
X 3 : Identify the category of each Android application (e.g. Business, Finance, Social, etc.).

Implicit (M): tasks that involve further analysis of the explicit data to infer the results implicitly.

M 1 : Find applications with the same set of permissions.

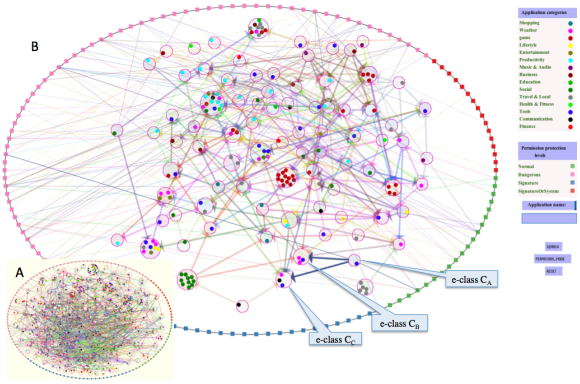

Figure 2: *Applying force-directed layout on our dataset. Inset A: the force-directed layout on the entire dataset. B: the layout of one-third of applications in our dataset (reducing the applications gives a better view of the layout).*

M 2 : Find the most frequently requested permissions.

M 3 : Identify any relationship among Android applications based on their requested permissions.

M 4 : Investigate how the gradual increase in permission usage changes the category of applications.

M 5 : Examine any correlation between the application categories and their permissions. This might assist the analyst to identify the expected permissions of each category.

4 Iterative Design Approach

We developed Papilio by means of iterative design. The explanation is annotated with task numbers where a specific feature was included to address a given task.

4.1 Apply Force-Directed Layout

Our first visualization of our dataset, inspired by TextArc [Pal02] and force-directed layout [FR91], is illustrated in Figure 2. It has four main components: the Android applications and their categories, permissions and their protection levels, the relations among applications and their permissions, and the relations among the application e-classes.

Applications and their categories: In Figure 2, Android applications requesting the same set of permissions are grouped together as an equivalence class (e-class) (M 1). Each e-class is depicted by a circle that contains a set of applications. Every single application is represented by a small circle, which is coloured according to its category (X 3). The legend in the upper right corner shows the mapping between categories and their colors. In Figure 2, for instance, C_A is an e-class containing one application from the Tools category.

Application permissions and their protection levels: In Figure 2, the Android permissions are each indicated by a small square placed along the large external ellipse. These permission squares are coloured by their protection level (X 2). Normal, dangerous, signature and signatureOrSystem permissions are coloured with green, pink, blue and red, respectively. Also, permissions with the same protection level are grouped together. Permissions within each group are positioned sequentially on the ellipse boundary.

Relation among applications and their permissions: Each permission in the external ellipse is connected by an undirected line to each e-class that uses it (X 1).

Relations among application e-classes: By considering the permissions used by sets of e-classes, it is apparent that the relations among the e-classes are partially ordered (M 3). In Figure 2, this parent-child relationship among e-classes is displayed via a directed line from the child e-class to its parent. A direct line among two e-classes means that the permission set of the child e-class is a superset of its parent's permission set. For instance, Figure 2 shows that there are parent-child relations between C_A (as a child e-class) and C_B and C_C (as parent e-classes) which implies that each permission set of C_B and C_C is a subset of the C_A permission set.

Critique of the first iteration: Using Force-Directed layout resulted in distributing the e-classes rather uniformly over the screen space. This has the advantage that all e-classes can be displayed at once which makes it easier to compare the e-classes. However, the directed and undirected lines connecting e-classes to each other and to permissions are also spread over the whole screen space. This resulted in many link crossings which in turn led to the familiar difficulties of many lines in dense graphs. However, as there exists an e-class in our data with an empty permission set, we are dealing with a rooted DAG. For our next iteration, we thought to make use of these hierarchical relationships within our data.

4.2 Taking Advantage of Hierarchy

Our second visualization, was inspired by the layered graph-layout style [BM01] to better display the hierarchical structure of our data. In this layout, a layer is assigned to each e-class (according to the *longest path layering* algorithm [RDM*87]) and e-classes belonging to the same layer are drawn in a horizontal row. Then the child-parent links are

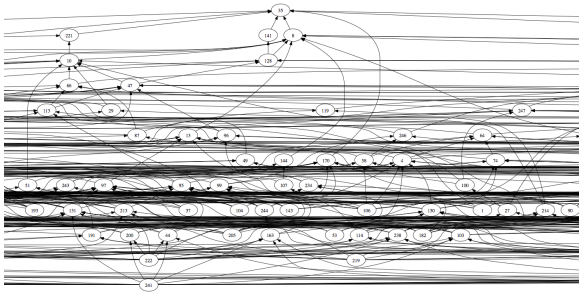

Figure 3: *The representation of e-classes and their relations in our dataset using the "dot" program. Many link crossings and long links has led to an unreadable representation.*

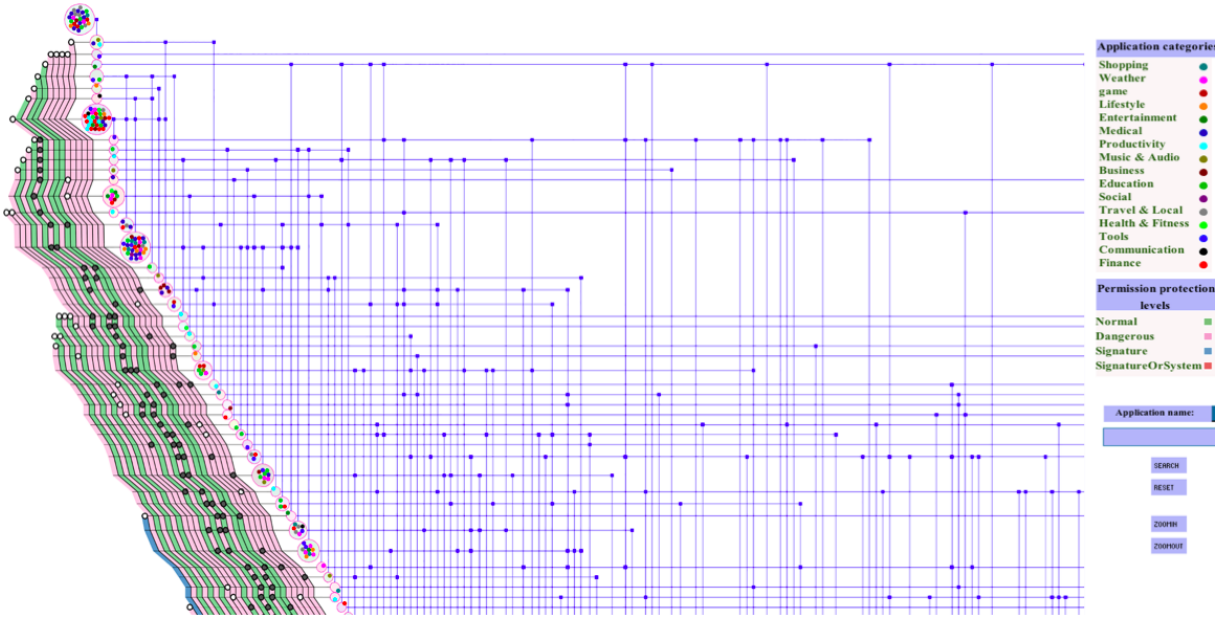

Figure 4: *Papilio is a two-sided visualization showing ordered application e-classes through the center, with the parent-child relations among e-classes on the right side and the requested permissions of e-classes on the left side.*

directed upward, connect e-classes of the successive or non-successive layers. To order the e-classes in each layer such that the link crossings between layers are minimized, the algorithm proposed in [GKNV93] is used as implemented in the *dot* program [Gra13]. Figure 3 shows the layout of application e-classes and their relations visualized by the *dot* program without the representation of permissions.

Critique of the second iteration: The main benefit of this visualization is its ability to demonstrate the hierarchical structure of our data. This may be reasonable for a small number of nodes and links. However, as the number of nodes or the link density increases, the representation becomes unreadable and hard to explore [GFC05]. This is even more problematic when visualizing skip links where the source and destination nodes belong to two non-successive layers.

Part of the problem of trying to make use of the hierarchy, is that our data is not a true hierarchy but a partial order and that we have two distinct types of edges to portray. In our final visualization Papilio, described in the next section, we work with both of these factors.

5 Papilio Visualization

We present Papilio as a new layout that minimizes the link crossings. The name Papilio means butterfly in Latin and was chosen because of the central insect-like body of application e-classes with two sides, though each has a radically different layout.

In exploring how to improve our e-classes relationship edges layout, we considered both node-link and matrix techniques. In Papilio we used a combination of matrix and node-link diagrams working towards taking advantage of the strengths of both techniques. For this approach, we were inspired by GeneaQuilts [BDF*10]. We have created Papilio as a two-sided visualization technique, where we visualize the hierarchical structure of the data on one side and the e-class permissions on the other side. Figure 4 illustrates the visualization of our dataset using Papilio.

Applications and their categories: In Papilio, an e-class groups together the applications requesting the same set of permissions (M 1). Within one e-class, each application is represented by a small circle colored according to its category (X 3), with the legend located at the right upper corner.

Relations among application e-classes: To explain the parent-child relationship between e-classes, consider C_A as an e-class, which contains all applications with exactly the same set of permissions. C_B, is a "child" of C_A, if it uses a superset of permissions (all of C_A's permissions plus some others). Thus, a given e-class can have many children. Also, e-class C_C is a sibling of C_B if C_B and C_C have exactly the same parents.

In Papilio, we visualize the parent-child relations on the right side of e-classes (M 3). To position the e-classes on the screen, we first assign a layer to each of them according to the *longest path layering* algorithm [RDM*87]. This layering algorithm allows us to assign the sibling e-classes to the same layer. This will help us in compressing the number of links to be drawn on the layout which will be discussed later in this section. To order e-classes within each layer, we adapt

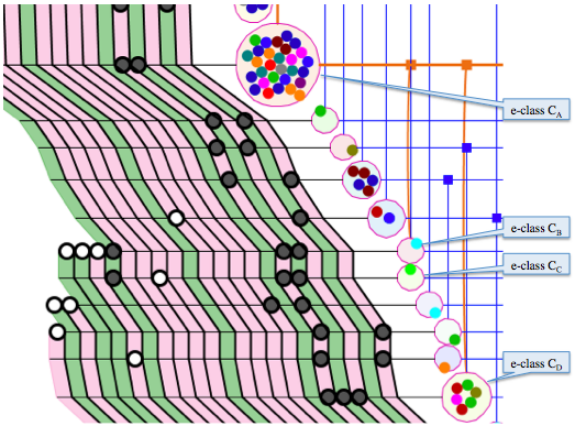

Figure 5: *C_A is a parent e-class for C_B, C_C and C_D. Also C_B and C_C are siblings which are recognizable based on their positions.*

a variant of the barycenter heuristic [SM05] which is implemented in the *dot* program [Gra13]. This ordering helps us to minimize the link crossings by positioning the connected e-classes close to one another. Using the layered e-classes, we translated the relations among e-classes to the *dot* format to compute the (X,Y) e-class positions. Starting from the topmost layer, our layout algorithm orders the e-classes in each layer according to their X positions obtained from running the *dot* program, and lays them out in order across the diagonal of a matrix. As much as possible we compress the distance between adjacent e-classes so that the visualization is of manageable size. We do ensure that we can draw both a vertical upward line and a horizontal line at the right side of each e-class. These lines are the links connecting e-classes based on the parent-child relations.

A vertical line reaching up from an e-class implies at least one parent and thus will contain at least one filled blue square. These filled blue squares directly above each e-class indicate the paths to its parent e-classes. The filled blue squares on the horizontal line to the right side of each e-class indicate the paths to its child e-classes. Each parent or child can be reached by following the corresponding blue squares to the left or bottom, respectively, until an e-class is hit. This can be seen in Figure 5 where e-class C_A is the parent of e-classes C_B, C_C and C_D.

For compressing the number of links to be drawn in our layout, we group sibling e-classes together and use e-class position as a visual element to display their sibling relation. To this end, if an e-class has siblings, all of them are located right below each other and just one vertical upward line connects them all to their common parents (Figure 5). This technique clearly demonstrates the sibling e-classes, and compresses the number of needed links among e-classes.

Application permissions and their protection levels: Each permission has a distinct connected representation starting at the first appearance of an e-class that makes use of this permission. Every permission connection is filled with a color representing its protection level (X 2). Normal, dangerous, signature and signatureOrSystem permission connections are filled with green, pink, blue and red, respectively.

This method of visualizing e-class permissions in Papilio allows the observer to infer some approximate statistical knowledge about the permission usage in our dataset. An example is discovering the set of most frequent requested permissions (M 2).

Relations among applications and their permissions: In Figure 4, the set of filled and unfilled circles on the horizontal line on the left side of each e-class, called the permission-line, indicate the e-class's permission set (X 1). The filled gray circles represent the permissions inherited from one of the e-class's parents. While unfilled ones are the permissions the application has in addition to its parents' permissions.

5.1 Papilio Interactions

Papilio visualization can be explored via the traditional zoom, pan and select. Zooming and panning offer the ability to explore the whole information space. Selecting either an application or a permission displays detailed information about the chosen application or permission. We added some additional exploration options.

Interaction with e-classes: In Papilio, it is important to be able to focus on an e-class and navigate through the parent-child relations to find e-classes that are connected to it. Thus, selecting an e-class highlights all the right hand side paths connecting the chosen e-class to its parents and children (Figure 6). Furthermore, the highlighted horizontal lines attached to the parents and the highlighted vertical lines attached to the children start strumming in order to ease the detection of the related e-classes [SNDC10].

Slide on Halo: In Papilio some points of interest such as the child and parent e-classes of a particular e-class might fall into the off-screen space. Spatial cognition of the off-screen e-classes as well as following the paths to reach them

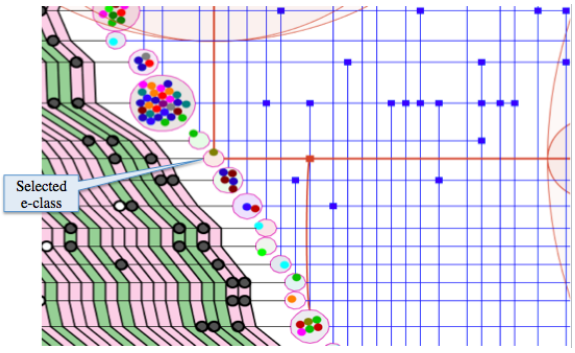

Figure 6: *Parents and children of a selected e-class. Two off-screen children and two off-screen parents are represented using Halos.*

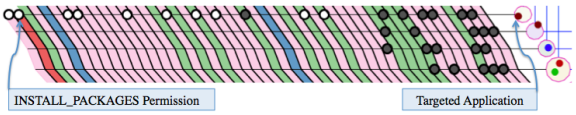

Figure 7: *An application requested INSTALL_PACKAGES permission which is not intended to be used by third-party applications.*

using pan and zoom techniques, can be a hard task, especially with large datasets [BR03,KEC06]. Thus, we designed Slide on Halo as a combination of Halo [BR03] and Bring & Slide [BDF*10] techniques. Slide on Halo addresses the spatial cognition of off-screen parent or child e-classes and also provides a smooth navigation towards them.

By selecting an e-class in Papilio, each of its off-screen child or parent e-classes is represented by means of an arc, henceforth called a Halo, on the right or top border of the display area, respectively. In fact, each Halo is part of a ring with a radius proportional to the distance of the targeted e-class from the display area. For instance in Figure 6, the chosen e-class has three children and two parents. However, just one child is visible on the screen and the rest child and parent e-classes are represented by Halos on the right and above the selected e-class, respectively.

Selecting a particular Halo, smoothly pans the current view both vertically and horizontally until the e-class corresponding to the selected Halo is reached.

6 Demonstrating the Utility of Papilio

From careful exploration and analysis of Papilio, we can discover how Android permissions are being requested by applications in practice. The results reported in this paper are specific to the 400 applications contained in our dataset. However, since we considered only the popular applications, we think our results can effectively explain the permission usage behaviour of popular applications that individuals commonly use. The following is a list of our findings from analyzing our dataset using Papilio.

Requesting standard permissions with signature or signaturOrSystem protection levels by some third-party applications: According to the Android documents [Goo12], some of the standard permissions with signature or signatureOrSystem protection levels are not intended to be used by third-party applications. Examples include CLEAR_APP_USER_DATA that allows an application to clear the user data and INSTALL_PACKAGES permission allowing an application to install packages. However, exploring Papilio reveals that there are a number of third-party applications requesting these types of permissions.

As an example, the indicated application in Figure 7 is requesting the INSTALL_PACKAGES permission. This application has more than 5 million installs according to Google Play statistics.

Little correlation between categories defined by Google and application permission requests: By exploring Papilio and going down the hierarchy, we found that most e-classes are small and contain only a few applications. Generally, this means that applications tend to request permissions independently of each other. However, there also exist a few large e-classes. At the top of the hierarchy, most of these large e-classes contain applications from several different categories. Having a large e-class at the top of the hierarchy with various categories is reasonable for e-classes that have general permissions such as the INTERNET. This is because applications from different categories might solely request this general permission. However, our observation shows that towards the middle and bottom portions of the hierarchy, fairly large e-classes consist of applications where either all or most of them are from the same category. Thus, it seems that there exists a small correlation between the application categories and their requested permissions when the number of requested permissions increases (M 4, M 5).

Despite this small correlation, any predominant appearance of a category within an e-class would be valuable for the security analyst in finding over-privileged applications. As a case in point, in Papilio there are two e-classes with only applications of the Game category. Therefore, any appearance of a Game application outside of these two e-classes would be interesting for further analysis (Figure 8).

No correlation between application categories of the parent and child e-classes: Navigating through the parent-child relations among e-classes in Papilio reflects that the categories of applications within a particular e-class do not correlate with the categories of applications in its parent or child e-classes. This means that, based on our dataset, the superset and subset relations among permission sets of two specific e-classes do not imply any correlation between the categories of applications they enclose (M 3, M 5).

If this correlation existed, then the analyst could refine the existing Google categorization into a hierarchical categorization conforming with the permission set hierarchy. This finer-grained categorization could realize the principle of least privilege. However, since we could not find any correlation between categories of parent and child e-classes, defining the finer-grained categorization based on permissions is dubious.

Particular application categories tend to request more (or less) permissions in comparison with other categories: In general, applications appearing towards the bot-

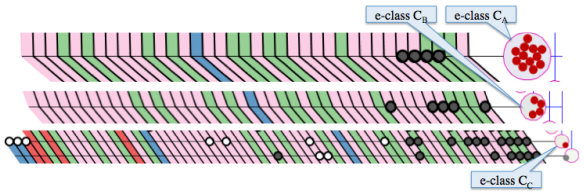

Figure 8: *Game applications grouped in C_A and C_B. The application in C_C is an outlier, potentially interesting for a security analyst.*

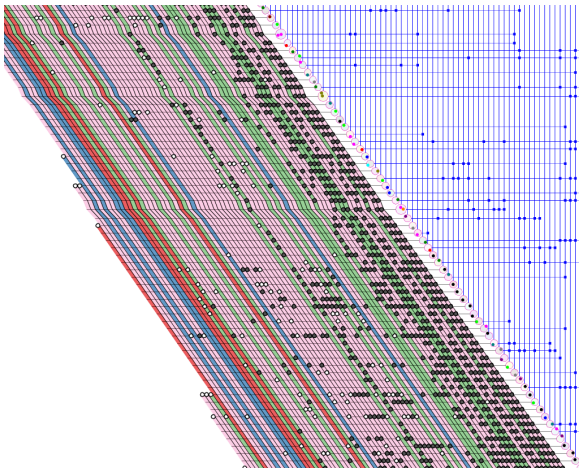

Figure 9: *Out of 130 permissions, just a small set of them are frequently requested by applications and the rest are either never requested or requested infrequently.*

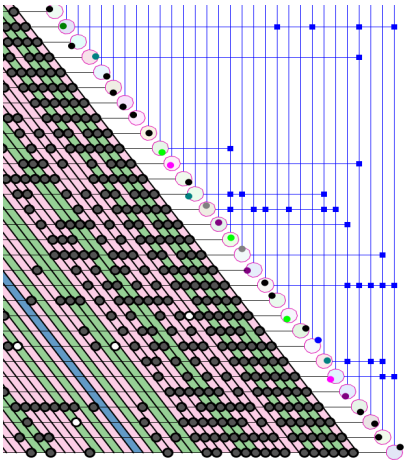

Figure 10: *15 out of 30 applications towards the bottom of the hierarchy belong to the Communication category (black colored applications).*

tom of the hierarchy tend to request more permissions than other applications. Consequently, if a category appears predominantly towards the bottom of Papilio, then we expect applications of that category request more permissions than other categories (M 4). As an example, by exploring Papilio, we found that applications belonging to the Communication category (represented by the black color) are mostly located at the bottom of the hierarchy. Thus, based on our dataset, we conclude that applications of the Communication category tend to request more permissions in comparison with the applications of other categories. Figure 10 shows part of the Papilio towards the bottom of its hierarchy in which 15 out of the 30 displayed applications are from the Communication category.

Similarly, applications emerging at the top of the hierarchy usually request less permissions in comparison with other applications. For instance, in Papilio applications of Medical category are mostly located at the top. Thus, we expect Medical applications to request less permissions compared to other categories.

Only a small set of permissions are frequently requested by applications: The permission usage in Papilio shows that only a small set of permissions are frequently requested by applications (Figure 9).

Also, we can see that out of 130 standard Android permissions in our dataset, a large number of them are either not requested by any application or requested very infrequently. This result agrees with Barrera *et al.* [BKvOS10]'s findings.

7 Conclusions and Future Work

In this paper, we presented Papilio, a novel visualization technique for exploring data about permission usage of real-world Android applications. E-classes in Papilio, each consisting of applications requesting the same set of permissions, form the boundary between the two sides of visualized data. The superset and subset relations among e-classes based on their permission set form a rooted DAG, visualized on one side of e-classes. Meanwhile, the permission sets of e-classes are visualized on the opposite side.

Exploring Papilio is possible through a set of interactions. Among these interactions, Slide on Halo is a technique for providing spatial awareness about off-screen e-classes as well as smooth navigation towards them. In this work, exploring and analyzing Papilio using its interactions led us to interesting security findings about how permissions are being requested by Android applications in practice.

Results from our research suggest some directions for future work. An important aspect of our visualization is providing knowledge regarding how permissions are being requested by applications. Since individual people are the ones who are in danger of installing applications that compromise their privacy, it would be nice to customize Papilio for personal mobile devices to give individuals an understanding of their privacy.

Other than Android, the Facebook platform and Google Chrome web browser are examples of systems using install-time permissions. Therefore, it is reasonable to conjecture that employing Papilio on datasets of Facebook applications or Chrome extensions could lead to novel findings regarding permission usage in those platforms. Furthermore, exploring the application of Papilio in other domains for visualizing the superset/subset relations among many-valued entities is left as future work. Examples of such datasets include disease and their symptoms, or food nutritional facts.

Finally, a natural future work is to evaluate the strengths and shortcomings of Papilio by conducting a study.

Acknowledgment

This research was supported in part by AITF, NSERC, GRAND, Surfnet, and SMART Technologies.

References

[Ama13] AMAZON: Amazon appstore for android. http://www.amazon.com/mobile-apps/b?node=2350149011, 2013. 3

[App12] APPBRAIN: Appbrain stats. http://www.appbrain.com/stats/, 2012. 1

[App13] APPLE: App review. https://developer.apple.com/ appstore/guidelines.html, 2013. 2

[BDF*10] BEZERIANOS A., DRAGICEVIC P., FEKETE J.-D., BAE J., WATSON B.: Geneaquilts: A system for exploring large genealogies. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1073–1081. 3, 6, 8

[BJL*13] BALEBAKO R., JUNG J., LU W., CRANOR L. F., NGUYEN C.: Little brothers watching you: Raising awareness of data leaks on smartphones. In *SOUPS* (2013). 2

[BKvOS10] BARRERA D., KAYACIK H. G., VAN OORSCHOT P. C., SOMAYAJI A.: A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM CCS* (2010), pp. 73–84. 2, 9

[BM01] BASTERT O., MATUSZEWSKI C.: Layered drawings of digraphs. In *Drawing Graphs*, vol. 2025 of *LNCS*. Springer Berlin Heidelberg, 2001, pp. 87–120. 5

[BR03] BAUDISCH P., ROSENHOLTZ R.: Halo: a technique for visualizing off-screen objects. In *Proceedings of the ACM SIGCHI* (2003), pp. 481–488. 8

[BW11] BAE J., WATSON B.: Developing and evaluating quilts for the depiction of large layered graphs. *Visualization and Computer Graphics, IEEE Transactions on 17* (2011), 2268–2275. 3

[DWvW12] DINKLA K., WESTENBERG M. A., VAN WIJK J. J.: Compressed adjacency matrices: Untangling gene regulatory networks. *TVCG 18*, 12 (2012), 2457–2466. 3

[EV10] EKLUND P., VILLERD J.: A survey of hybrid representations of concept lattices in conceptual knowledge processing. In *Proceedings of the 8th ICFCA* (2010), Springer, pp. 296–311. 3

[FCH*11] FELT A. P., CHIN E., HANNA S., SONG D., WAGNER D.: Android permissions demystified. In *Proceedings of the 18th ACM CCS* (2011), pp. 627–638. 2

[FR91] FRUCHTERMAN T. M., REINGOLD E. M.: Graph drawing by force-directed placement. *Software: Practice and experience 21*, 11 (1991), 1129–1164. 5

[FVWSN08] FEKETE J.-D., VAN WIJK J. J., STASKO J. T., NORTH C.: The value of information visualization. In *Information Visualization*. Springer, 2008, pp. 1–18. 2

[GFC04] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: A comparison of the readability of graphs using node-link and matrix-based representations. In *InfoVis* (2004), pp. 17–24. 3

[GFC05] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization 4*, 2 (2005), 114–135. 6

[GKNV93] GANSNER E. R., KOUTSOFIOS E., NORTH S. C., VO K.-P.: A technique for drawing directed graphs. *Software Engineering, IEEE Transactions on 19*, 3 (1993), 214–230. 6

[Goo12] GOOGLE: Android permissions. http://developer. android.com/reference/android/Manifest. permission.html, 2012. 8

[Goo13a] GOOGLE: Android market. http://www.play.google .com/store, 2013. 3

[Goo13b] GOOGLE: Android permission protection levels. http://developer.android.com/ guide/topics/manifest/ permission-element.html, Oct 2013. 3

[Gra13] GRAPHVIZ: Graphviz - graph visualization software. http://graphviz.org, 2013. 6, 7

[HFM07] HENRY N., FEKETE J.-D., MCGUFFIN M. J.: Nodetrix: a hybrid visualization of social networks. *Visualization and Computer Graphics 13*, 6 (2007), 1302–1309. 3

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *Visualization and Comp Graphics* (2000), 24–43. 3

[KEC06] KELLER R., ECKERT C. M., CLARKSON P. J.: Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? *InfoVis 5* (2006), 62–76. 8

[KSH01] KOHONEN T., SCHROEDER M. R., HUANG T. S. (Eds.): *Self-Organizing Maps*, 3rd ed. Springer, 2001. 2

[McA12] MCAFEE: Android malware promises video while stealing contacts. https://blogs.mcafee.com/mcafee-labs/android -malware-promises-video-while-stealing -contacts, 2012. 2

[MDNST08] MESSAI N., DEVIGNES M.-D., NAPOLI A., SMAIL-TABBONE M.: Many-valued concept lattices for conceptual clustering and information retrieval. In *Proceedings of 18th ECAI* (Amsterdam, 2008), IOS Press, pp. 127–131. 3

[Pal02] PALEY W. B.: Textarc: Showing word frequency and distribution in text. In *IEEE InfoVis* (2002), vol. 2002. 5

[Pan13] PANDASECURITY: Eeki.a. http://www.pandasecurity .com/homeusers/security-info/215107/Eeki.A, 2013. 2

[PNK11] PUPYREV S., NACHMANSON L., KAUFMANN M.: Improving layered graph layouts with edge bundling. In *Graph Drawing* (2011), Springer, pp. 329–340. 3

[RDM*87] ROWE L. A., DAVIS M., MESSINGER E., MEYER C., SPIRAKIS C., TUAN A.: A browser for directed graphs. *Software: Practice and Experience 17*, 1 (1987), 61–76. 5, 6

[SM05] SIIRTOLA H., MÄKINEN E.: Constructing and reconstructing the reorderable matrix. *InfoVis 4*, 1 (2005), 32–48. 7

[SNDC10] SCHMIDT S., NACENTA M. A., DACHSELT R., CARPENDALE S.: A set of multi-touch graph interaction techniques. In *ACM ITS* (2010), ACM, pp. 113–116. 7

[SS75] SALTZER J. H., SCHROEDER M. D.: The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (1975), 1278–1308. 2

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Trans on Systems, Man and Cybernetics* (1981), 109–125. 3

[Sym13] SYMANTEC: Android geinimi. http://www.symantec .com/security-response/ writeup.jspdocid 5403-99, 2013. 2

[TPP09] TAMASSIA R., PALAZZI B., PAPAMANTHOU C.: Graph drawing for security visualization. In *Graph Drawing* (2009), Springer, pp. 2–13. 2

[VLKS*11] VON LANDESBERGER T., KUIJPER A., SCHRECK T., KOHLHAMMER J., VAN WIJK J. J., FEKETE J.-D., FELLNER D. W.: Visual analysis of large graphs: State-of-the-art and future research challenges. In *Computer graphics forum* (2011), vol. 30, Wiley Online Library, pp. 1719–1749. 3

[WBS*07] WATSON B., BRINK D., STALLMANN M., DEVARAJAN R., RAKOW M., RHYNE T.-M., PATEL H.: Visualizing very large layered graphs with quilts. In *IEEE Information Visualization Conference Poster* (2007). 3

[Wil92] WILLE R.: Concept lattices and conceptual knowledge systems. *Computers & Mathematics with Applications 23* (1992), 493–515. 3