Compartmentation Policies for Android Apps: A Combinatorial Optimization Approach

Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez

Department of Computer Science, Universidad Carlos III de Madrid, Spain
guillermo.suarez.tangil@uc3m.es**, {jestevez, pperis}@inf.uc3m.es

Abstract. Some smartphone platforms such as Android have a distinctive message passing system that allows for sophisticated interactions among app components, both within and across app boundaries. This gives rise to various security and privacy risks, including not only intentional collusion attacks via permission re-delegation but also inadvertent disclosure of information and service misuse through confused deputy attacks. In this paper, we revisit the perils of app coexistence in the same platform and propose a risk mitigation mechanism based on segregating apps into isolated groups following classical security compartmentation principles. Compartments can be implemented using lightweight approaches such as Inter-Component Communication (ICC) firewalling or through virtualization, effectively fencing off each group of apps. We then leverage recent works on quantified risk metrics for Android apps to couch compartmentation as a combinatorial optimization problem akin to the classical bin packing or knapsack problems. We study a number of simple yet effective numerical optimization heuristics, showing that very good compartmentation solutions can be obtained for the problem sizes expected in current's mobile environments.

Key words: Smartphone security, permission based security, malware, collusion attacks, risk assessment.

1 Introduction

Android's security model is substantially different from that of standard desktop operating systems, as it was designed to better fit the architecture and intended usage of smartphones. The device is seen as a platform with a number of available services, such as storage, networking, and a collection of sensors [1]. Access to each service is provided through a system API freely available to apps yet restricted with a permission system. Thus, an app must request the appropriate permission in its manifest in order to gain access to protected API calls. Permissions can be also used by apps to control interactions among components, for instance by specifying which privileges a caller must have in order to use a component. Recent studies by Felt et al. on the effectiveness of permission systems in smartphone platforms conclude that they are quite effective at protecting users [2]. However, in the case of Android it has been pointed out that apps often

** Current email address: guillermo.suarez-tangil@rhul.ac.uk request a significant amount of permissions identified as potentially dangerous. This exposes users to frequent warnings, which drastically reduces effectiveness.

Android apps are considered mutually distrusted and are isolated from each other. Thus, each app has its own process and can access its own data only. Despite this isolation, Android provides the developer with a rich inter-application message passing system. This pursues several goals, including to facilitate component reuse and inter-application collaboration. In Android, developers are encouraged to leverage existing data and services offered by other apps, which is achieved by dividing an app into components and then exchanging information within the app boundaries—ICC, or Inter-Component Communication—and across applications—IPC, or Inter-Process Communication. This is mostly achieved through *intents*, which can be thought of as messages that allow implicit and explicit communication among components.

The Perils of Coexistence. The Android app interaction model creates numerous security risks [3]. A careless developer may accidentally expose functionality that another (malicious) app can exploit to, for instance, trick it into performing an undesirable action. Thus, vulnerable apps can unintentionally provide attackers an interface to privileged resources in what is known as a *confused deputy attack*. Additionally, in a compromised device messages exchanged between two components could be intercepted, stopped, and/or replaced by others, as they are generally not encrypted or authenticated.

Deliberate collusion attacks are not only possible but also quite simple to implement. Two or more malicious apps can cooperate to violate security policies in the so-called *permission re-delegation attacks* [4]. Permission re-delegation takes place when an app with sufficient permissions performs a privileged task that is requested by another app that does not have those permissions, which effectively undermines the user-approved permission system. To further complicate matters, a sophisticated attacker might not even rely on the IPC/ICC subsystem, but on covert channels as a substitute to the official communication interface. This would provide colluding apps with an alternative—and most likely unnoticed—vehicle to exchange information. Chandra et al. [5] have recently conducted a comprehensive study of such covert channels in smartphones, showing that they abound and that some of them offer reasonable bandwidth. More generally, apps with networking privileges can also use communication channels external to the smartphone to exchange information.

Related Work. Countering attacks that exploit inter-app communications is a challenging task [6]. Bugiel et al. introduced in [7] a framework called TrustDroid to separate trusted from untrusted applications into domains, firewalling ICC messages among domains. Partly based on this concept, Samsung has recently released the KNOX Container [8], so that apps and data inside a container are isolated from apps outside it. According to Samsung, KNOX is intended to facilitate the coexistence of work and personal content on the same device, this being a more lightweight solution than using a separate virtual machine (VM) for each compartment. Interested readers can find in [9] an overview of recent progress in virtualization techniques for mobile systems.

Monitoring and enforcing restrictions on app interactions has been a major research theme almost since the first Android releases. Dietz et al. introduce in [10] *Quire*, a signature-based scheme that allows developers to specify local (ICC) and remote (RPC) communication restrictions. Other proposals such as TaintDroid [11], AppFence [12], or XManDroid [13] closely monitor apps to enforce given security policies. While the first two use dynamic taint analysis to prevent data leakage and protect user's privacy, the latter extends Android's security architecture to prevent privilege escalation attacks at runtime.

Motivation and Contribution. A common theme in all the solutions discussed above is that app segregation is ultimately driven by a user-defined policy. But delegating such a burden to users can only result in a very limited protection, since policy making is unanimously recognized as a difficult task. Users hardly understand the repercussions of granting an app a given set of permissions, let alone those of all possible combinations of apps. Furthermore, policies will likely be user- and even context-specific, so a one-size-fits-all approach does not seem a sensible choice either. Lastly, the problem of leveraging covert channels for command, control, and communication among colluding apps is yet to be addressed. Isolation is generally recognized as one of the most economic and effective ways to dismantle covert channels, but then again it is unclear which apps should be set apart from which others without totally disrupting the very purpose of inter-app communication.

In a related but different area, risk analysis techniques such as [14–18] have recently gained much attention as attractive mechanisms to effectively signal potential threats and better communicate them to final users. Most of such techniques are essentially based on deriving a numerical score from various app features, generally its permissions. Motivated by the discussion provided above, in this paper we make the following contributions:

1. We argue that current risk assessment schemes based on examining apps in isolation can only offer a limited vision of the actual risk, since they fail to model the perils of app coexistence in security models such as that of Android. Ideally, risk assessment should be redefined to extend its scope to all apps residing in the platform, possibly considering dynamic contextual variables too. To formalize this, we introduce an *Unrestricted Collusion (UC)* model that captures these points in a very simple way.
2. We reuse existing risk scoring techniques and adapt them to the UC model. Simple experiments with typical apps show that, for instance, as little as 10 apps may pose a risk level higher than that the risk obtained for 75% of well-known malware instances.
3. We then revisit the classical idea of risk mitigation through compartmentation (as in, e.g., the Brewer-Nash model [19]), a notion that has been used for decades both in corporations and by the intelligence community, and is implicit in some of the works that have addressed the problem of app isolation. However, considering the complexities and limitations that policy making entails, we use quantified risk metrics to formulate the problem as a class of mathematical optimization problems known as *packing problems.*

This addresses the compartmentation problem in a very effective way while reducing user involvement to a bare minimum.

4. We explore 14 heuristics for two practical settings—risk minimization given a fixed number of compartments per mobile terminal, and minimization of the number of compartments given a maximum tolerable risk level for each of them. Our experimental results show that the problem is practically tractable for the sizes involved in current mobile's environments.
5. Lastly, we introduce a freely available online service called DroidSack that implements app compartmentation as introduced in this paper.

2 A Quantified Risk Model for App Colocation

2.1 Risk Scoring Functions for Individual Apps

Several proposals have recently addressed the design of mechanisms to palliate the ineffective way in which permissions are used to communicate potential risks to the user [14]. Wang et al. introduced in [20] DroidRisk, a permission-based quantitative risk assessment metric for Android apps. DroidRisk draws inspiration from standard methods in quantitative risk assessment and associates with each app $\boldsymbol{a}$ the risk quantity

$$R(\boldsymbol{a}) = \sum_i R(p_i) = \sum_i L(p_i) I(p_i), \tag{1}$$

where $R(p_i)$ is the risk level of permission p_i, $L(p_i)$ and $I(p_i)$ are the likelihood and the impact of permission p_i, respectively, and the sum is taken over all requested permissions. The likelihoods $L(p_i)$ are empirically estimated by applying Bayes' rule to a dataset of benign and malicious apps. As for the impacts $I(p_i)$, they are set to 1 for normal permissions and to 1.5 for dangerous ones. These values are also empirically chosen so as to maximize discrimination between goodware and malware.

Similar mechanisms are presented by Peng et al. in [16] and then further explored in an extended version of that paper by Gates et al. in [17]. Here the authors develop various risk scoring functions also based on the set of permissions an app requests, including probabilistic generative models such as Basic Naive Bayes (BNB), Naive Bayes with informative Priors (PNB), Mixture of Naive Bayes (MNB), and Hierarchical Mixture of Naive Bayes (HMNB). The work presented in [17] also explores a related approach in which the rarity of permissions—computed as the logarithm of the associated probability—is used to construct risk metrics. Each app $\boldsymbol{a}$ is modeled as a pair $a = [c_i, \boldsymbol{x_i} = (x_{i,1}, \ldots, x_{i,M})]$, where c_i is the category of the app, M is the number of permissions, and $\boldsymbol{x_i}$ a binary vector indicating which permissions the app requests. Two risk metrics are introduced. The first is called the Rarity Based Risk Score (RS) and associates with each app the number

$$RS(\boldsymbol{x_i}) = \sum_{m=1}^{M} x_{i,m} \cdot \ln\left(\frac{N}{c_m}\right), \tag{2}$$

where N is the total number of apps. A variant called Rarity Based Risk Score with Scaling (RSS) is also explored. It uses scaling factors w_n to penalize high risk permissions more than low risk ones

$$RRS(\boldsymbol{x_i}) = \sum_{m=1}^{M} x_{i,m} \cdot w_m \cdot \ln\left(\frac{N}{c_m}\right). \tag{3}$$

Other proposals, such as the work reported in [18, 21, 15], introduce more complex risk assessment mechanisms and consider factors other than permissions, such as *intents* or the presence of native code, among others.

2.2 Extending Risk Scoring to App Compartments

Feature-based risk scores. Essentially all the risk scoring mechanisms proposed so far represent an app $\boldsymbol{a}$ as a feature set

$$\boldsymbol{a} \longleftrightarrow \phi_{\boldsymbol{a}} = \{f_1, \dots, f_M\}, \tag{4}$$

where each f_i is a feature associated with a particular *risk factor*, i.e., an aspect of the app which is relevant when measuring the risk it poses. Permissions are, by far, the most common risk factors considered by existing risk assessment metrics. Thus, most risk metrics represent $\phi_{\boldsymbol{a}}$ as a binary vector in which a one in the i-th position means that the app requests permission p_i, and vice versa.

Risk quantification is effectively done by some scoring function $\rho(\boldsymbol{a})$ returning, in general, a positive real number proportional to the amount of risk posed by $\boldsymbol{a}$. As discussed in [16], it is reasonable to assume that risk scoring functions are *monotonic*. In our feature-based framework, monotonocity for a risk scoring function ρ means that, if $\phi_{\boldsymbol{a}}$ and $\phi_{\boldsymbol{b}}$ are the feature sets associated with apps $\boldsymbol{a}$ and $\boldsymbol{b}$, then

$$\phi_{\boldsymbol{a}} \subseteq \phi_{\boldsymbol{b}} \Rightarrow \rho(\boldsymbol{a}) \leq \rho(\boldsymbol{b}). \tag{5}$$

That is, adding risk factors to an app does not decrease risk.

The Unrestricted Collusion (UC) model. We now consider the problem of measuring the risk of a set of apps $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_N\}$ running on the same device. This strongly depends on the particular platform used. We will assume a rather permissive co-existence model such as the one provided by Android, in which collusion is facilitated not only by side channels, but also directly (re-delegation attacks) and indirectly (confused deputy attacks) by the IPC subsystem. Thus, we define an Unrestricted Collusion (UC) model as follows: in terms of risk, a set of apps running on the same platform can be viewed as a single app whose risk factors are the union of the risk factors of the constituent apps. More formally:

$$\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_N\} \longleftrightarrow \phi_{\boldsymbol{a}_1, \dots, \boldsymbol{a}_N} = \bigcup_{i=1}^{N} \phi_{\boldsymbol{a}_i}. \tag{6}$$

In practical terms, the UC model states that apps can communicate with each other without restrictions. Thus, if one of them has been granted permission to access a particular resource, all of them can also access that resource via the first app. We believe the UC model is reasonable for the current smartphone ecosystem dominated by social, gaming, and sport apps that are increasingly supporting resource sharing and other forms of interactions with each other.

Abusing notation $\rho(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\})$ can be computed by any feature-based risk scoring function for individual apps by just applying it to the union of all feature sets of the integrating apps. A priori, it is unclear what the relationship between $\rho(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\})$ and $\{\rho(\boldsymbol{a}_1), \ldots, \rho(\boldsymbol{a}_N)\}$ should be. For example, both DroidRisk—expression (1)—, RS—expression (2)—, and RSS—expression (3)— are "sublinear", in the sense that they have the subadditivity property

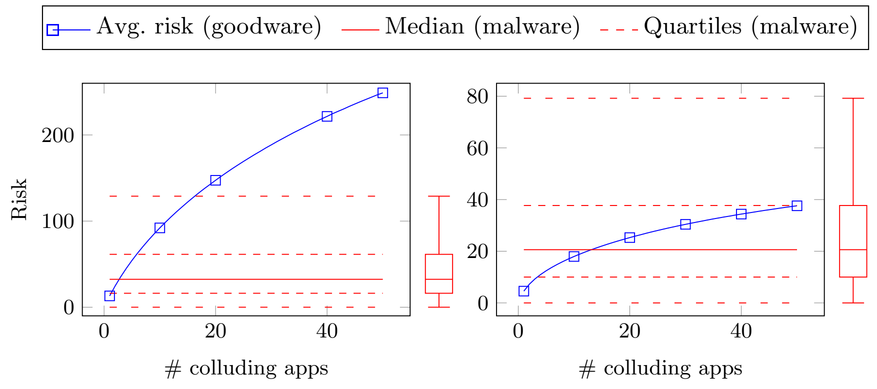
$$\rho(\bigcup_{i=1}^{N} \boldsymbol{a}_i) \leq \sum_{i=1}^{N} \rho(\boldsymbol{a}_i). \tag{7}$$

Note that this will not generally hold for nonlinear risk scoring, e.g, those based on subsets of risk factors. Sublinear risk scoring functions are relevant for a class of compartmentation heuristics developed later in Section 3.

2.3 An Empirical Analysis of Colocation Risk

We first conducted an empirical evaluation of the risk metrics discussed above in order to assess the risk of colocated apps in the UC model. We implemented DroidRisk [20] and the RS and RSS metrics proposed in [17]. In all cases, parameters were estimated using a dataset composed of over 15K apps from the Google Play market and over 15K malicious apps from VirusShare. Each app in the dataset was preprocessed and transformed into its corresponding feature vector with the permission-related information so as to train each risk model. Overall, our results show risks distributions very similar to those reported in the original papers and confirm that permission-based risk metrics offer a fair degree of discrimination between goodware and malware.

We next considered the case of a platform hosting $N \in \{10, 20, 30, 40, 50\}$ colluding apps and measured the risk of the entire group. The choice of this range is motivated by a 2014 report from Nielsen establishing that smartphone owners use between 20 and 30 apps on a regular basis [22]. For each value of N, we randomly selected a group of apps from our dataset of non-malicious apps, computed the risk of the set, and repeated the experiment 1000 times. Fig. 1 shows the risk using DroidRisk and RS/RSS as underlying risk metrics. For a better understanding of the implications of colluding attacks in terms of quantified risk, each plot is accompanied by the risk distribution of malware. The results suggest that, for instance, just 10 apps pose a risk level higher than 75% of the malware according to the RSS metric. The figures vary when using DroidRisk, yet the fundamentals are the same. Overall, as the number of apps increases so does the risk. As expected, risk growth slows down after certain number of apps since the likelihood of acquiring more risk factors for the group (e.g., additional permissions) gets lower.



(a) RSS (b) DroidRisk

Fig. 1: Quantified risk of collusion for different number of apps

3 Optimal Risk Compartmentation Policies

3.1 Two Compartmentation Problems

Our scheme relies on a quantified risk-driven compartmentation policy. That means that, in principle, there are no predefined conflict of interest classes, as in the classical Brewer-Nash model, nor any other enforceable mandatory controls. Contrarily, compartmentation is implemented by computing the risk of a group of apps coexisting in the same compartment and checking whether this is acceptable or not. Note that, in doing so, compartmentation takes place with minimal user intervention.

We consider two different settings in which app compartmentation can occur. The first one, called the RISKPACK problem, models a scenario in which it is feasible to define a notion of a *maximum tolerable risk*, this being an upper bound to the risk that each compartment can assume. For simplicity, we assume that all compartments have the same risk capacity. This can be straightforwardly extended to the general case in which the user can define compartments with different risk bounds.

Definition 1 (RISKPACK). *Given a set $\boldsymbol{A}$ of N apps, for each $\boldsymbol{S} \subseteq \boldsymbol{A}$ a risk measure $\rho(\boldsymbol{S}) \in \mathbb{Z}^+$, a finite set $\boldsymbol{K}$ of N compartments, and a maximum tolerable risk $\tau \in \mathbb{Z}^+$ common to all compartments $k \in \boldsymbol{K}$, the RISKPACK problem is to find an integer number of compartments Z and a Z-partition $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_Z$ of the set $\boldsymbol{A}$ such that $\rho(\boldsymbol{S}_i) \leq \tau$ for all $i = 1, \ldots, Z$. A solution is said to be optimal if it has a minimal Z.*

Note that in RISKPACK the number of available compartments is equal to the number of apps, and it is implicitly assumed that $\rho(\boldsymbol{a}) \leq \tau$ for all $\boldsymbol{a} \in \boldsymbol{A}$.

That is, all apps will be eventually assigned to a compartment, the challenge being how to use the minimum number of compartments while not exceeding the risk capacity in none of them.

The second problem, called RISKMIN, is intended for a more practical situation, in the following sense. On the one hand, the semantics of the risk scoring functions are often unclear. Quantified risk approaches are generally not intended as final indicators to be communicated to users, but rather as intermediate variables to be used in a higher lever decision making process. Thus, in RISKMIN the focus is not on each compartment's risk value in absolute terms but rather on minimizing it. On the other hand, instead of assuming that a pool of as many as necessary compartments is available, we assume a fixed, and possibly small, number of them. This is more commensurate with current smartphones' capabilities, since it is unrealistic to assume they will soon be able to support a substantial number of virtual machines.

Definition 2 (RISKMIN). *Given a set $\boldsymbol{A}$ of N apps, for each $\boldsymbol{S} \subseteq \boldsymbol{A}$ a risk measure $\rho(\boldsymbol{S}) \in \mathbb{Z}^+$, and a finite set $\boldsymbol{K}$ of $M \leq N$ compartments, the RISKMIN problem is to find a Z-partition $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_Z$ of the set $\boldsymbol{A}$ such that $\sum_{i=1}^{Z} \rho(\boldsymbol{S}_i)$ is minimal. Other target functions are possible, for example minimizing $\max_i \rho(\boldsymbol{S}_i)$.*

For simplicity, a detailed discussion on how to introduce restrictions such as these in our model is left out of this paper. Nonetheless, we anticipate that most problem solving strategies would be able to deal with them straightforwardly.

Online vs Offline Packing. As in the case of many classical packing problems, it seems reasonable to consider two versions in which compartmentation can take place. In the *online* setting, apps must be installed in a compartment one at a time, without considering which the next app(s) would be. Contrarily, in an *offline* setting all apps are given upfront. It is easy to prove that the online problem is more difficult and that there is no algorithm that always gets the optimal solution.

3.2 Complexity Analysis and Heuristics

RISKPACK is a variant of the combinatorial optimization Bin-Packing Problem (BPP) [23]. There is, however one significant difference: while in BPP the space in a bin occupied by two objects is the sum of their sizes, in RISKPACK there might not be an straightforward relationship between the risk of two apps put together and the risks of each one of them isolated. When the risk scoring function is sublinear, RISKPACK reduces to the VM (Virtual Machine) packing problem recently explored by Sindelar et al. in [24], in which several virtual machines jointly packed in a server share memory pages and, therefore, occupy less space than the sum of their individual sizes.

The RISKMIN problem is a variant of the Multiple Subset Sum (MSS) problem, and can be also seen as a Multiple Knapsack Problem (KPP) if compartments with different risk tolerances are assumed [23]. As in the case of RISKPACK, the key difference is that risk aggregation by the scoring function might not be additive. As discussed in the next section, this can negatively impact the ability to develop efficient approximations.

Both RISKPACK and RISKMIN are NP-hard since they contain the BPP and the MSS/MKP, respectively, as special cases when the risk scoring function is strictly additive in the sense that

$$\rho(\bigcup_{i=1}^{N} \boldsymbol{a}_i) = \sum_{i=1}^{N} \rho(\boldsymbol{a}_i). \tag{8}$$

Furthermore, a direct reformulation of the results provided in [24] determines that, for arbitrary (i.e., nonlinear) risk scoring function, RISKMIN is hard to approximate, whereas for the case of RISKPACK the question is open. Despite this hardness result, we will later see that standard heuristic strategies attain sufficiently good solutions for many instances that arise in practice.

Even though BPP and MSS/MKP are known to be NP-hard, excellent solutions to large instances can be obtained by relatively simple algorithms. Many heuristics have been developed for both problems, often resulting in solvers that provide fast but generally suboptimal solutions. We have adapted some of those heuristics to our risk packing/minimization problems, and also explored others commonly used in minimization problems. In total, we explored 14 different heuristics. A short description of each of them is provided in Table 1. The implementation in all cases is straightforward.

4 Experimentation

4.1 RISKPACK

We first obtained a risk score model of our dataset for RS, RSS, and DroidRisk as described in Section 2. Then, we computed the number of compartments required to fit $N \in \{10, 30, 50\}$ apps given a *maximum tolerable risk* $\tau \in [0, 1]$. In all the risk metrics used it is possible to compute the maximum attainable risk for a set of apps. This allows us to express risk as a percentage relative to the maximum, which is arguably a more understandable communication instrument for users. For each number of compartments N, we randomly selected a group of apps from our dataset of non-malicious apps, repeating the experiment 1000 times, testing all heuristics described in Table 1 for the selected group of apps. Due to space restrictions, Fig. 2 only shows the results reported using RSS and DroidRisk (DR). RS yields very similar results to RSS.

Results show that apps can be segregated into a small number of compartments with a very low risk tolerance each. For instance, 10 apps require just 2-4 compartments for a maximum risk tolerance of 1%-2%, increasing to around

	Heuristic	**Description**
RISKPACK	NF	*Next Fit.* When processing the next app, see if it fits in the same compartment as the last app. Start a new empty compartment if it does not.
	FF	*First Fit.* As NF but rather than checking just the last compartment, check all previous compartments.
	BF	*Best Fit.* Place the app in the tightest compartment, i.e., in the spot so that the smallest residual risk is left.
	CF	*Cheapest Fit.* Place the app in the compartment in which it causes the lowest risk increment.
	FFD	*First Fit Decreasing.* Offline analog of FF. Sort the apps in decreasing order of risk and then apply FF.
	BFD	*Best Fit Decreasing.* Offline analog of BF. Sort the apps in decreasing order of risk and then apply BF.
	CFD	*Cheapest Fit Decreasing.* Offline analog of CF. Sort the apps in decreasing order of risk and then apply CFD.
RISKMIN	HC	*Hill Climbing.* Start with a random assignment of apps to compartments. Pick one app randomly and move it to a randomly chosen compartment. Keep it there if the overall risk decreases; otherwise undo the move. Repeat until no improvement is achieved for L consecutive moves.
	MR	*Minimum Risk.* Place the app in the compartment with minimum risk.
	$B^\star$	*Best Risk.* Place the app in an empty compartment, if any. Otherwise, place it in a compartment in which it causes no risk increment, if possible. Otherwise, place it where it causes the highest risk increment.
	$CR^\star$	*Cheapest Risk.* Place the app in an empty compartment, if any. Otherwise, place it in a compartment in which it causes no risk increment, if possible. Otherwise, place it where it causes the lowest risk increment.
	$MRD^\star$	*Minimum Risk Decreasing.* Place the app in an empty compartment, if any. Otherwise, place it in a compartment in which it causes no risk increment, if possible. Otherwise, place it in the compartment with lowest risk.
	$BRD^\star$	*Best Risk Decreasing**. Offline analog of $B^\star$. Sort the apps in decreasing order and then apply $B^\star$.
	$CRD^\star$	*Cheapest Risk Decreasing**. Offline analog of $C^\star$. Sort the apps in decreasing order and then apply $C^\star$.

Table 1: Heuristics for the RISKPACK and RISKMIN problems.

10 compartments for 50 apps. Interestingly, the risk vs. number of compartment relation is not linear: while a massive risk reduction can be done with just 2 or 3 compartments, further reducing the risk translates into an exponential increase on the number of compartments. Additionally, note that the number of compartments strongly depends on the risk metric used. For instance, a user that could only afford a 2% of the overall risk when installing 10 apps requires around 2 compartments with RSS but over 8 with DroidRisk. Some heuristics such as FF and FFD consistently outperform across settings and in all scenarios. These heuristics are known to behave well in classical combinatorial packing problems, so this does not come as a surprise.

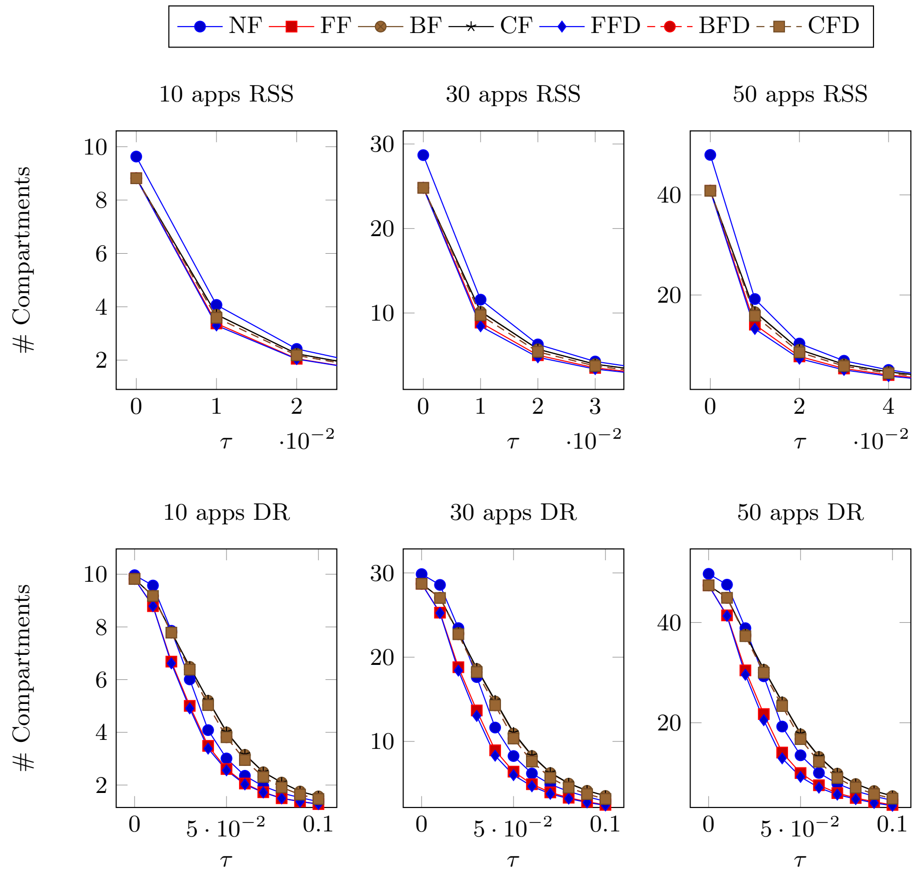


Fig. 2: Solutions for the RISKPACK problem using different heuristics: number of compartments used for a maximum tolerable risk τ.

4.2 RISKMIN

We tested RISKMIN solvers using the same experimental setting described above. Based on this, we computed the risk reported by the platform given a fixed amount of compartments. For the sake of simplicity, we assume that the risk of the platform is determined by the compartment with higher risk. Fig. 3 shows the results obtained using RSS and DroidRisk (DR). As in the case of RISKPACK, the overall risk can be effectively minimized even when using a small number of compartments. From all heuristics tested, MRD, MR, and HC perform better than the others. Note the duality among the curves in Figs. 2 and 3. Unlike RISKPACK, however, the performance of the different heuristics varies significantly. For instance, MRD* achieves a risk of 10^{-2} with 30 apps and 10 compartments in DroidRisk (DR), while BR* attains about 0.13 in the same setting.

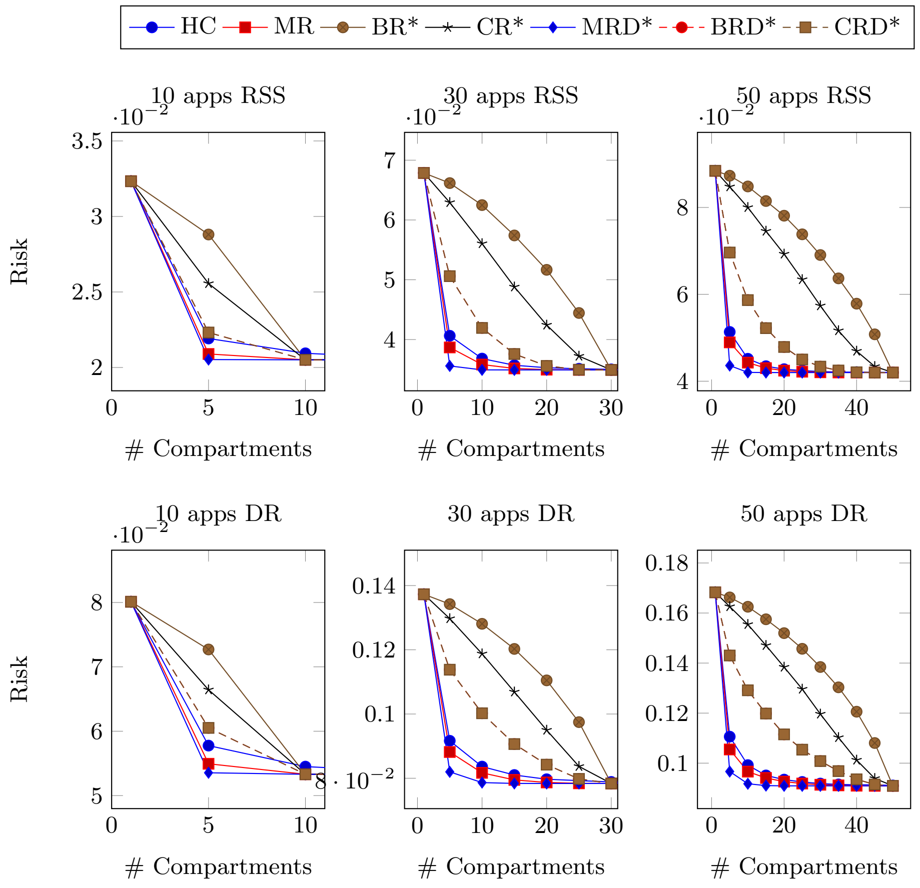


Fig. 3: Solutions for the RISKMIN problem using different heuristics: residual risk vs. number of available compartments.

4.3 DroidSack: An Online Compartmentation Service

We have implemented a freely available online service called DroidSack that offers app compartmentation as introduced in this paper. The service is exposed through a REST HTTP-based API publicly available[1]. The API comprises two services, `GET RISKPACK` and `GET RISKMIN`, which implement solvers for the two problems. In the current version, apps are provided through their full names from the Google Play market. The service connects to the market, retrieves the app, and extract the manifest to compute the risk. Candidate solutions are returned as JSON objects. Detailed instructions on how to use it along with a basic HTML interface for manual usage are provided in the URL given above.

[1] `http://www.seg.inf.uc3m.es/DroidSack`

5 Conclusions and Future Work

In this paper, we have revisited the security problems derived from app coexistence in mobile platforms such as Android. To counter them, we have adopted a compartmentation approach driven by a quantified risk assessment metric. We have introduced a collusion model that facilitates extending existing risk metrics for smartphone apps to sets of apps. We have then posed two combinatorial optimization problems for two practical settings and discussed our experimental results with simple yet effective numerical optimization heuristics. Overall, our results suggest that very good compartmentation solutions can be obtained quite efficiently for the sizes expected in current's mobile environments.

Our proposal presents a number of limitations that should be tackled in future work. For instance, we deliberately do not consider app collusion via Internet. We believe that, although a perfectly valid mechanism to share resources, this problem should be addressed by different means—e.g., by system-level monitoring and firewalling. Similarly, we have only considered permissions as the only type of risk factor, since it is the most common feature used by existing risk assessment metrics. However, considering additional aspects of an app such as IPC calls might provide a more precise assessment of the risk and should be further studied. Finally, dynamic reallocation policies, as opposed to re-solving the problem again with different inputs, might be worth-exploring. For example, such reallocation policies would be interesting when using context-driven risk measures in which the risk of an app changes as the context varies.

Acknowledgments

We are very grateful to the anonymous reviewers for constructive feedback and insightful suggestions. This work was supported by the MINECO grant TIN2013-46469-R (SPINY: Security and Privacy in the Internet of You) and the CAM grant S2013/ICE-3095 (CIBERDINE: Cybersecurity, Data, and Risks).

References

1. Suarez-Tangil, G., Tapiador, J.E., Peris, P., Ribagorda, A.: Evolution, detection and analysis of malware for smart devices. IEEE Communications Surveys & Tutorials **16**(2) (May 2014) 961–987
2. Felt, A.P., Greenwood, K., Wagner, D.: The effectiveness of application permissions. In: USENIX Web application development. WebApps'11 (2011) 7–7
3. Chin, E., Felt, A., Greenwood, K., Wagner, D.: Analyzing inter-application communication in android. In: Mobile sys., apps., and services, ACM (2011) 239–252
4. Felt, A., Wang, H., Moshchuk, A., Hanna, S., Chin, E.: Permission re-delegation: Attacks and defenses. In: USENIX Security Symposium. (2011) 1–16
5. Chandra, S., Lin, Z., Kundu, A., Khan, L.: Towards a systematic study of the covert channel attacks in smartphones. Technical report, Univ. of Texas (2014)
6. Fang, Z., Han, W., Li, Y.: Permission based android security: Issues and countermeasures. Computers & Security **43** (2014) 205–218

7. Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.R., Shastry, B.: Practical and lightweight domain isolation on android. In: Security and privacy in smartphones and mobile devices. SPSM '11, New York, ACM (2011) 51–62
8. Samsung: White paper: An overview of samsung knox (April 2013) `http://www.samsung.com/es/business-images/resource/white-paper/2014/02/Samsung\_KNOX\_whitepaper-0.pdf`.
9. Jaramillo, D., Furht, B., Agarwal, A.: Mobile virtualization technologies. In: Virtualization Techniques for Mobile Systems. Springer (2014) 5–20
10. Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., Wallach, D.S.: Quire: Lightweight provenance for smart phone operating systems. In: USENIX Security. (2011) 16
11. Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., Sheth, A.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: USENIX OS design and implementation. (2010) 1–6
12. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: Computer and communications security, ACM (2011) 639–652
13. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.: Xmandroid: A new android evolution to mitigate privilege escalation attacks. Technical report, Technische Universitat Darmstadt (2011)
14. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. SPSM '11, NY, USA (2011) 3–14
15. Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X.: Riskranker: scalable and accurate zero-day android malware detection. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM (2012) 281–294
16. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Using probabilistic generative models for ranking risks of android apps. In: Computer and communications security, ACM (2012) 241–252
17. Gates, C., Li, N., Peng, H., Sarma, B., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Generating summary risk scores for mobile applications. Dependable and Secure Computing, IEEE Transactions on **11**(3) (May 2014) 238–251
18. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in android applications for malicious application detection. Information Forensics and Security, IEEE Transactions on **9**(11) (Nov 2014) 1869–1882
19. Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: IEEE Symposium on Security and Privacy, Oakland, CA, USA. (1989) 206–214
20. Wang, Y., Zheng, J., Sun, C., Mukkamala, S.: Quantitative security risk assessment of android permissions and applications. In: Data and Applications Security and Privacy. DBSec'13, Springer-Verlag (2013) 226–241
21. Chakradeo, S., Reaves, B., Traynor, P., Enck, W.: Mast: Triage for market-scale mobile malware analysis. In: Security and Privacy in Wireless and Mobile Networks. WiSec '13, NY, USA, ACM (2013) 13–24
22. Nielsen: Smartphones: so many apps, so much time. Available Online (Last visited October 2014) (July 2014)
23. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. J. Wiley & Sons (1990)
24. Sindelar, M., Sitaraman, R.K., Shenoy, P.J.: Sharing-aware algorithms for virtual machine colocation. In: ACM Symposium on Parallelism in Algorithms and Architectures. (2011) 367–378