Fencing Off Apps for Fun and Hygiene

Juan E. Tapiador

jestevez@inf.uc3m.es

Universidad Carlos III de Madrid, Spain

Talk based on the paper:
"Compartmentation Policies for Android Apps: A Combinatorial Optimization Approach."
G. Suarez-Tangil, J.E. Tapiador, P. Peris-Lopez.
Proc. 9th Intl. Conf. Network and Systems Security (NSS 2015)

City University London, November 26, 2015

Android Security Model

- App sandboxing at the process and file levels.
  - But supports shared UIDs, used extensively by system applications
- Permissions
  - Control access to the platform services and resources
  - All-or-nothing (soon to change)
- Code signing
  - Used to establish trust relationships between apps
  - Same origin policy in app updates
- Modified SELinux
  - Policy enforcement is only applied to core system daemons
  - Apps run in permissive mode (violations are logged but do not cause runtime errors)

Quantifying App Risk

- Are permissions effective to communicate potential risks to the user? [Felt et al., 2011]
- **Risk scoring functions for individual apps**
  - DroidRisk [Wang et al., 2013]

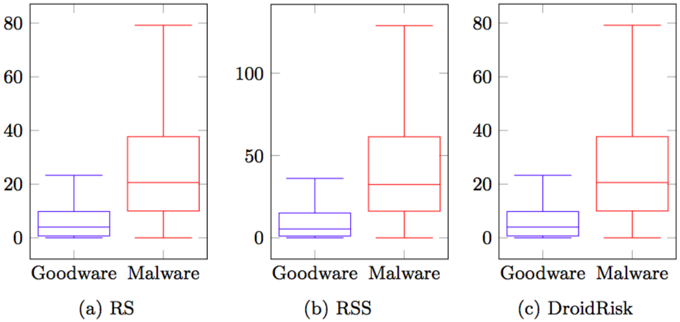
$$R(a) = \sum_i R(p_i) = \sum_i L(p_i) I(p_i)$$

  - Rarity-based Risk Scores [Gates et al., 2014]

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

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

Quantifying App Risk



- Evaluation with our dataset:
  - 15K goodware apps from Google Play
  - 15K malware apps from VirusShare

The Problem of Malicious Information Flows

- Root cause is that Android's info security model is too coarse grained.
  - Users grant apps the right to access sensitive info or services.
  - An app might need to legitimately access sensitive info, *but only for a specific limited purpose*. Android doesn't support such fine-grain policies.
  - One potential result is leakage of sensitive information.
  - Problematic even for apps that are not malicious as they may suffer from leaks through, e.g., advertisement libraries.
- Information-flow analysis:
  - Static approaches.
  - Goal is tracking sensitive info through the app by starting at pre-defined sources and following the data flow until it reaches a sink.
  - Using an accurate runtime execution model is critical, and this is particularly challenging for Android.

Information-Flow Analysis for Android

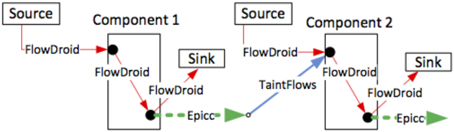
- Early efforts
  - Commercial: AppScan Source (IBM),Fortify SCA (HP).
  - Academic: SCanDroid, ScanDal, AndroidLeaks, CHEX, LeakMiner, Trustdroid.
  - **Weaknesses**: inaccurate analysis, imprecise Android API model.
- FlowDroid (2014):
  - Precise Context, Field, Object-sensitive and Lifecycle-aware
- EdgeMiner (2015):
  - Add-on to Flowdroid to better treat callbacks and indirect information flow transfers.
- DroidSafe (2015):
  - Not flow-sensitive (FlowDroid is) but far more complete model of Android runtime execution.

The Perils of App Coexistence

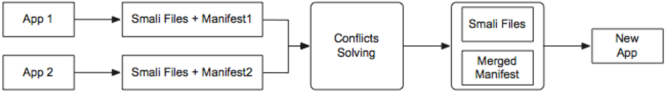
- Android encourages component reuse and inter-application collaboration:
  - Apps divided into components
  - Exchange information and leverage existing services within the app boundaries (ICC) and across applications (IPC)
    - Intents
    - Service binding
- Problem: **potentially insecure information flows across apps**:
  - Confused deputy attacks
  - Collusion attacks
  - Communication via covert channels
- **Plus other risks**: *activity hijacking*, *intent spoofing*, ...
- Current risk assessment schemes based on examining apps in isolation can only offer a limited vision of the actual risk

Extending info-flow analysis to app sets

- Epicc (2013): tool to resolve Intent destinations.
- IccTA (2014), DidFail (2014-15): FlowDroid + Epicc



- ApkCombiner (2015): Merges 2 apps replacing IAC by ICC



Motivation and Overview

- Countering attacks that exploit inter-app communication:
  - ICC / IPC firewalling
  - Samsung KNOX container
  - Virtualization (maybe soon)
- All require user-defined app segregation policies
  - Security policy making is difficult and error prone
  - User- and context-dependant policies
- **Our work:**
  - Formalize adversarial model and extend risk scoring functions to app sets
  - Risk mitigation through compartmentation, with policies formulated as solutions to an optimization problem
  - Online (free) compartmentation service
  - WIP: extension using risk metrics derived from joint info-flow analysis.

Extending Risk Scoring to App Sets

- **Feature-based risk scores**
  - An app $\mathbf{a}$ is modeled as a feature set

$$\mathbf{a} \mapsto \phi_{\mathbf{a}} = \{f_1, \ldots, f_M\}$$

  - Each f_i is a risk factor (generally permissions)
  - Risk factors can also be contextual and time dependant

$$\mathbf{a}(t) \mapsto \phi_{\mathbf{a}}(t) = \{f_1, \ldots, f_M, c_1(t), \ldots, c_L(t)\}$$

- **Risk scoring function**: $\rho(\mathbf{a}) \geq 0$
  - Generally monotonic:

$$\phi_{\mathbf{a}} \subseteq \phi_{\mathbf{b}} \Rightarrow \rho(\mathbf{a}) \leq \rho(\mathbf{b})$$

(i.e., adding risk factors does not decrease risk)

Extending Risk Scoring to App Sets

- **The Unrestricted Collusion (UC) model**
  - Worst-case scenario:

    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.

- **Risk scoring**

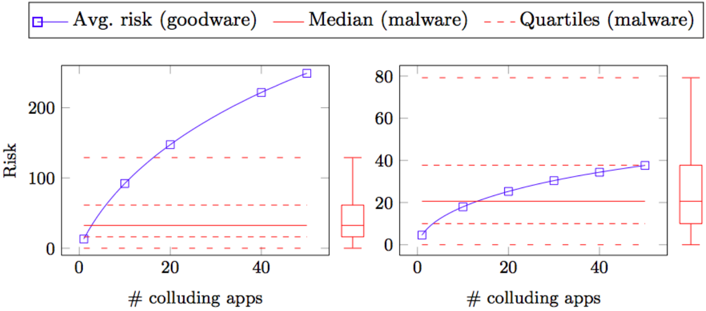
$$\mathbf{S} = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\} \mapsto \phi_{\mathbf{S}} = \bigcup_{i=1}^{N} \phi_{\mathbf{a}_i}$$

and

$$\rho(\mathbf{S}) = \rho(\phi_{\mathbf{S}})$$

An Empirical Analysis of Colocation Risk

- Quantified risk of collusion for different number of apps:
  - Sets of $N \in \{10, 20, 30, 40, 50\}$ colluding apps
  - Measure the risk of the entire group



(a) RSS

(b) DroidRisk

Optimal Risk Compartmentation Policies

- Compartmentation policies as in the classical Brewer-Nash model, but using quantified risk measures instead of predefined mandatory controls (minimal user intervention).
- Two compartmentation problems. Intuitively:
- RISKPACK:
  - It's feasible to define a notion of *maximum tolerable risk*—an upper bound for the risk each compartment can assume. All compartments have the same risk capacity.
  - No limit to the number of compartments.
- MINRISK:
  - Fixed, and often small, number of compartments.
  - The semantics of the risk scoring function are unclear, so the focus is not on the risk value in absolute terms but rather on minimizing it.
- *Online* vs *offline* compartmentation.

Optimal Risk Compartmentation Policies

Definition (RISKPACK). *Given:*

- *a set* $\mathbf{A}$ *of* N *apps*
- *for each* $\mathbf{S} \subseteq \mathbf{A}$ *a risk measure* $\rho(\mathbf{S}) \in \mathbb{Z}^+$
- *a finite set* $\mathbf{K}$ *of* N *compartments, and*
- *a maximum tolerable risk* $\tau \in \mathbb{Z}^+$ *common to all compartments* $k \in \mathbf{K}$*,*

the RISKPACK problem is to find an integer number of compartments Z *and a* Z*-partition* $\mathbf{S}_1, \ldots, \mathbf{S}_Z$ *of the set* $\mathbf{A}$ *such that* $\rho(\mathbf{S}_i) \leq \tau$ *for all* $i = 1, \ldots, Z$*. A solution is said to be optimal if it has a minimal* Z*.*

Optimal Risk Compartmentation Policies

Definition (RISKMIN). *Given:*

- *a set* $\mathbf{A}$ *of* N *apps*
- *for each* $\mathbf{S} \subseteq \mathbf{A}$ *a risk measure* $\rho(\mathbf{S}) \in \mathbb{Z}^+$*, and*
- *a finite set* $\mathbf{K}$ *of* $M \leq N$ *compartments,*

the RISKMIN problem is to find a Z*-partition* $\mathbf{S}_1, \ldots, \mathbf{S}_Z$ *of the set* $\mathbf{A}$ *such that* $\sum_{i=1}^{Z} \rho(\mathbf{S}_i)$ *is minimal. Other target functions are possible, for example minimizing* $\max_i \rho(\mathbf{S}_i)$.

Complexity

- Both RISKPACK and RISKMIN are NP-hard.
- RISKPACK is a variant of the Bin-Packing Problem (BPP)
  - One important difference: while in BPP $size(\{a, b\}) = size(a) + size(b)$, in RISKPACK there might not be an straightforward relationship between $\rho(\{a, b\})$ and $\rho(a)$ and $\rho(b)$.
  - If ρ is sublinear, RISKPACK reduces to the recently proposed VM-Packing problem
- RISKMIN is a variant of the Multiple Subset Sum (MSS) problem and can be also seen as a Multiple Knapsack Problem (MKP).
  - Again, the key difference is that risk aggregation by the scoring function might not be additive.

Heuristics

- Developed for BPP and MSS/MKP and adapted to RISKPACK and RISKMIN

	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.

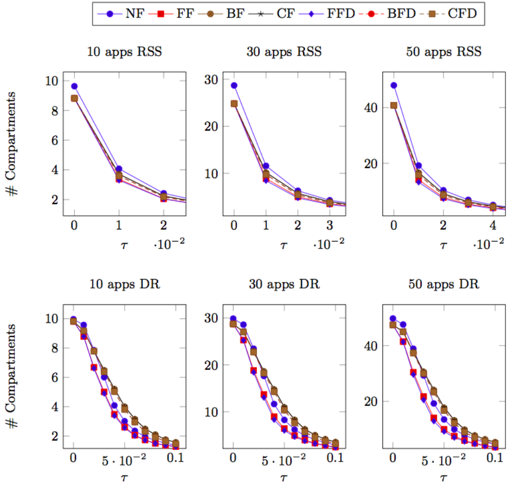
Heuristics

	Heuristic	Description
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*	*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*	*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*	*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*	*Best Risk Decreasing**. Offline analog of B*. Sort the apps in decreasing order and then apply B*.
	CRD*	*Cheapest Risk Decreasing**. Offline analog of C*. Sort the apps in decreasing order and then apply C*.

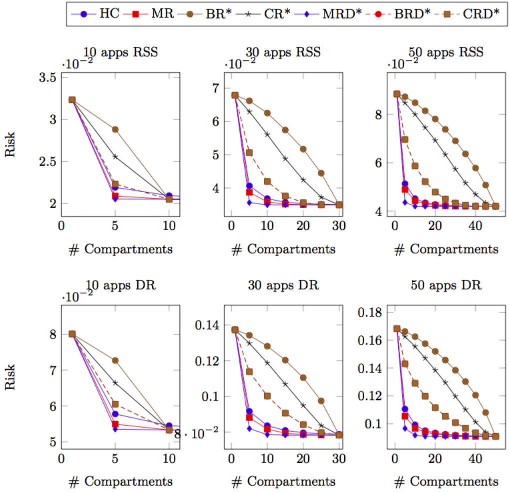
Experimentation

- RISKPACK
  - Computed number of compartments required to fit $N \in \{10, 30, 50\}$ apps given $\tau \in [0, 1]$.
- RISKMIN
  - Computed the risk (the higher across all compartments) obtained to fit $N \in \{10, 30, 50\}$ apps given a fixed number of compartments.
- Only non-malicious apps.
- Each experiment averaged over 1000 runs.
- Only RSS and DroidRisk. (RS behaves very similarly to RSS.)

Experimentation: RISKPACK



Experimentation: RISKMIN



DroidSack: An Online Compartmentation Service

- Implements heuristics solutions to user-defined RISKPACK and RISKMIN instances.
- Service exposed through a REST HTTP-based API:
  - `GET RISKPACK`
  - `GET RISKMIN`
- Currently apps are provided through their full names in the Google Play market.
- Solutions are returned as JSON objects.
- Freely available at:

  `http://www.seg.inf.uc3m.es/DroidSack`

Concluding Remarks

- App collusion via Internet is deliberately not considered.
- Dynamic reallocation policies (e.g., context driven or after installing new app / updating).
- Restricted compartmentation:
  - Mutually exclusive apps. Subsets of apps that, either because of external policy or personal privacy preferences, should not coexist in the same compartment.
  - Category segregation. Akin to apps but with categories.
  - User-defined groups, i.e., personal categories.
- Risk factors other than permisions, particularly info flow analysis (DidFail, ApkCombiner).

Q & A

Thank you!