View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Bournemouth University Research Online

# Malevolent App Pairs: An Android Permission Overpassing Scheme

Antonios Dimitriadis
Dept. Electrical and Computer Engineering, Democritus University of Thrace, Greece
andimitr@ee.duth.gr

Pavlos S. Efraimidis
Dept. Electrical and Computer Engineering, Democritus University of Thrace, Greece
pefraimi@ee.duth.gr

Vasilios Katos
Dept. of Computing & Informatics
Bournemouth University, UK
vkatos@bournemouth.ac.uk

## ABSTRACT

Portable smart devices potentially store a wealth of information of personal data, making them attractive targets for data exfiltration attacks. Permission based schemes are core security controls for reducing privacy and security risks. In this paper we demonstrate that current permission schemes cannot effectively mitigate risks posed by covert channels. We show that a pair of apps with different permission settings may collude in order to effectively create a state where a union of their permissions is obtained, giving opportunities for leaking sensitive data, whilst keeping the leak potentially unnoticed. We then propose a solution for such attacks.

## Keywords

Android smartphones, privacy, data exfiltration, malevolent applications, covert channel

## 1. INTRODUCTION

The ever increasing number of available applications for smartphones and tablets, inevitably increases the risk of unintentional exposure and disclosure of user data to a large number of third parties. There is a plethora of applications available for immediate download and installation within a matter of minutes. Many of these applications collect personal data, some of which can be sensitive, biometrical, location-revealing or behavioural such as the user's browsing history [19]. In order to gain access to a certain user data attributes, an installed application needs to request the appropriate permission from the user. There are two fundamental assumptions on this authorisation model. First, the user is expected to understand the permissions and the consequences of granting them to the under installation application. This is problematic because individuals may lack the appropriate background and knowledge in order to make informed decisions on their privacy [3]. Such permissions-based mechanism is widely criticized by developers, marketers, and end users [3] for the high coarse-grained control of application permissions and cumbersome permissions management setting.

The second assumption is that the authorisation model is enforced. If an application requests a limited number of permissions, the user may consider this application less risky than some other application requiring more permissions. Based on this perception, the user may eventually allow its installation. For example, if an application does not ask for Internet access permissions, the user may expect that this application has no means to transfer any collected sensitive data outside the phone device to a third party. Such reasonable expectation does not hold in practice and the misconception of applications running in isolated environments with no collusion capabilities leads to distorted risk perceptions as the users are under the impression that they can protect their privacy soley on approval of application persmissions indepentently [13]. The aim of this paper is to empirically invalidate this assumption by demonstrating that applications can collaborate with the use of covert channels (CC).

In [14], a number of covert and overt communication channels were proposed, enabling applications to collude and therefore indirectly escalate their permissions. Furthermore, it was proposed that timing CC which require precise timing, need to synchronise using a new REQUIRE_PRECISE_TIMING permission [14]. In the same work, the authors state that privacy risks cannot be solely assessed by examining the application's permissions. Two seemingly benign applications having only one and different permission each, is adequate to introduce privacy failures if such applications co-exist in the same device. This is in line with the popular saying that "the resulting system is greater that the sum of its parts".

In this paper we consider a scheme that uses two seemingly unrelated participant apps for data theft and exfiltration. The app pair uses covert channels for communication concealment and for exfiltrating data to a third party. This app pair does not need precise timing, as the synchronization is made by using system commands with no need for separate permissions or events.

## 2. THE MALEVOLENT PAIR SCHEME

Our approach involves two different apps and the goal is to gain unauthorized access to user data in order to transmit these to a third party without the explicit permision and consent of the user. This will be achieved by siphoning the data from the network isolated app to the Internet permitted app, using a CC and more specifically a timing CC [10]. Overt communication, such as use of external or

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CF'16, May 16-19, 2016, Como, Italy*
© 2016 ACM. ISBN 978-1-4503-4128-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2903150.2911706

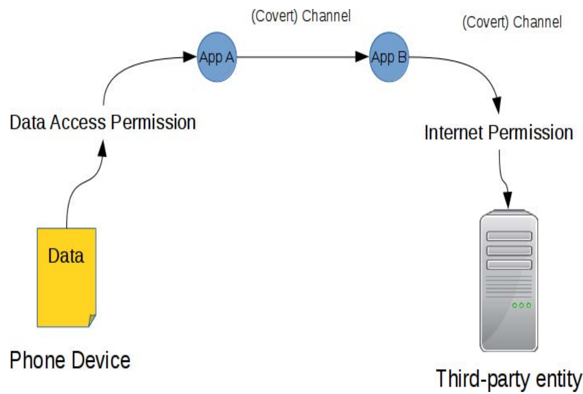


**Figure 1: Overview of the attack vector.**

internal storage staging of data is not considered as this can be easily detected. Fig. 1 illustrates this attack vector, with A and B being the two respective apps.

For the description of this attack we consider an Android device and two applications A and B. Application A is granted with only one permission, which gives access to some sensitive data field, and application B is granted only with the Internet access permission. Thus application A can manage data on the device, but cannot transmit them to a third party. On the other hand, application B can transmit data to a remote user, whilst having no access to any sensitive personal data. Our goal is to build a stealthy cooperative technique between A and B, allowing the smuggling of the sensitive data to a third party.

As it is suggested, for each data requested by an app (and, in general, any applications running on the user-side), it should be clearly stated if these data will be transmitted outside the smart device (the data item as it is or results obtained from this data item) or if they will only be used inside the smart device [18].

## 2.1 Covert Channel setup

The scheme proposed in this paper is based on the implementation of a hybrid CC. Hybrid channels use an indirect flow of information between the different classification domains, thus even a proactive system administrator may not be able to reason about all the effects that occur when actions take place within a system [17]. The goal of such a channel is not necessarily to obscure the data flowing through the channel, but to obscure the very fact that a channel exists [5].

In order to construct this CC we take advantage of two capabilities available in Android OS smartphone devices. Firstly, we exploit the battery percentage variable as a synchronization signal for apps A and B. The synchronization is achieved with periodical check of the battery level, as this does not need any specific permissions from the Android system. The second capability is the use of the method getRunningTasks, which also does not ask for any permission. This command informs an app on which apps are currently running and is executed periodically.

An important precaution taken in the attacking scheme is that the synchronization can only start if both apps are in the background at the same time. This condition is necessary because in order to execute the synchronization event the sending app needs to restart itself. If the app would be running, the user would notice the unexpected app activity.

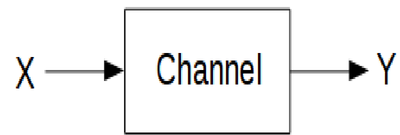


**Figure 2: Data transmission.**

Moreover, performing the synchronization only while the apps are in the background reduces the potential noise of the covert channel. Usually a channel is noisy. This means that, given a value for X, the resulting value for Y is not determined (illustrated in Fig. 2) but, instead, has a probability distribution, which is determined by the characteristics of the channel [15]. The outputs observed by the user Y may depend on the activities of users other than X. In Android systems each installed app is considered as a new user for the system [1], and thus we can correlate the users referred in [15] with the Android apps. We consider the sending attacking app (app A) as the user X, and the receiving attacking app (app B) as the user Y.

Interference to the actual attack process can be achieved by other apps or by the user himself. In [15], it is noted that when the sequence of all inputs from users other than user X during a trial is constant, the channel capacity takes on its maximum value of one bit per trial.

Thus, we firstly attempt to isolate the interference from the user. In order to achieve this, the attack starts only if the phone is not in use by the user; the relevant information can be retrieved by checking the screen state (via the function “isInteractive”). Moreover, other apps installed on an Android device do not start or stop the attacking apps which we used for our attack implementation, and thus we do not have interference of other apps. In order to verify this, we tested many different apps available on the Android App Store. Thus, the user X has no interference from other users, which makes the channel capacity reach its maximum value of one bit per trial.

The combination of these capabilities allow apps A and B to synchronize and communicate with no noticeable overheads or particular impact to the system. The two apps are not required to start simultaneously. As such, synchronization and data transmission are achieved with no required behavioural changes from the two apps; if app A stops by moving to the background $n$ seconds after a specific battery percentage change or value, this information would be available to app B.

A straightforward encoding scheme for the CC communication could be for example plain ASCII. If app A for instance wishes to send the message “Hello World” to app B, then A could remain open for 72 time units, and this would correspond to encoding the first character “H” which matches the decimal number of 72 in ASCII. Such time laps is captured by app B which will be decoding the 72 time units back to “H”. This example is illustrated in Fig. 3.

The process can be repeated for as long as it is required in order to send all private data. At the end of the transmission, the source app can send an EOF code to signal that the transmission is complete. Then B can move on to sending the received data to a third party.

Figure 3: Cooperation of processes A and B

## 2.2 The Data Exfiltration

The purpose of a CC is the concealment of the whole data transmission process - that is, in order for the CC to meet its goal, the covert communication will need to be applied end-to-end. Thus, an attack which aims to remain stealthy, needs to incorporate the CC logic not only between the app pair but also between the app B and the final destination. Against this background we introduce a second CC between the mobile device and the third party.

### 2.2.1 *Covert channel exfiltration*

Many apps designed to *legitimately* touch base with a server include synchronisation functionality which is used to beef up the user experience. This feature will be employed for staging the covert exfiltration of the acquired data by app B.

We assume that the first syncing attempt each day will be scheduled under a certain code match. App B and the server would have a prior agreement of the encoding scheme used for the covert communication. For simplicity, let us assume that this is based on standard ASCII encoding, similar to the CC between the two apps outlined above. Let us also assume, without loss of generality, that synchronisation is performed once a day. As both the server and the App have Internet connectivity, we can assume that they can obtain a fairly accurate timestamp. Of course we cannot assume absolute synchronisation between the two separate clocks. However, our scheme is able to tolerate discrepancies between them, at the cost of a lower capacity for the covert channel.

The server and app can use a commonly agreed starting time $t_0$ from which to measure an time offset. In our example, let $t_0$ be equal to midnight. The period $\Delta_{EOF} = t_{EOF} - t_0$ where $t_{EOF}$ is the shortest time lapse between $t_0$ and the synchronisation event and this can designate the completion of covert data transmission; any time greater than $\Delta_{EOF}$ would designate that there are more data sent by the CC and such data can be decoded by calculating the ASCII representation of the difference $t_1 - \Delta_{EOF}$, where $t_1$ is the timestamp of the initiation of the synchronisation process. In our example, if $\Delta_{EOF} = 5$ time units (say minutes), then if the synchonisation begins at $t_1 = 77$, the resulting character sent would be "H" (i.e. ASCII 72).

In the above example, the exfiltration would need $\|m\|$ days, where $m$ is the message to be covertly transmitted. Obviously, if synchronisation is performed more frequently, this would reduce the days required to complete the transmission.

## 3. IMPLEMENTATION OF THE PROPOSED ATTACK

As a proof of concept we implemented our scheme on the corresponding apks of the apps. More precisely, we used the ExDialer - Dialer & Contacts app published by the Modoohut account, and the BBC news app published by the BBC Worldwide (Ltd) account. These apps have been arbitrarily chosen as representative examples of popular apps having a fit for purpose combination of permissions. The ExDialer app is a dialer app which has access to the Device Id & call information, but has not access to the Internet. The BBC app has access to the Internet, but has not access to the Device Id & call information field. For the developed proof of concept the ExDialer app reads the phone number and transmits it to the BBC app, which finally transmits it to a third party.

We unpacked the apk[1] files of these apps and introduced the malevolent action of the proposed attack. It should be noted that the new apks are signed by our keys and not by the official vendor's keys, since the original signatures are not valid for the patched apk. This approach is used for illustration purposes. The beneficiary of such an attack scheme would either be the actual app vendor implementing the CC within the original app, or a third party adversary who will need to successfully obtain a valid signatur for the patched apk. For the proof of concept a local server was setup for harvesting the stolen private data.

It is assumed that the desired data that the modified apps wish to exfiltrate are one or more of the stored phone numbers. Knowing in advance the type of data to be exfiltrated, we can employ a more specific encoding scheme that supports only the ten digits:

$$x(n) = \begin{cases} [0,5) & \text{if } n = \text{EOF} \\ [5n+5, 5n+10) & \text{otherwise} \end{cases} \quad (1)$$

The above assigns each number to a 5ms window. The time required for each digit varies from 5ms to 55ms, which leads to a transfer rate of at least 18 digits per second for this hop of the communication. Even though in many cases the capacity of the covert channel is an important feature of an attack (for example, one might argue that a covert channel of bandwidth greater than 100bps would pose a significant threat to data security [16]), the malevolent pair attack focuses on the data exfiltration of specific sensitive data fields. The size of these data will in most cases be small. Moreover, there is no particular time bound on the duration of such an attack, as long as the attack can remain unnoticed.

[1] Android application package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware (https://en.wikipedia.org/wiki/Android_application_package).

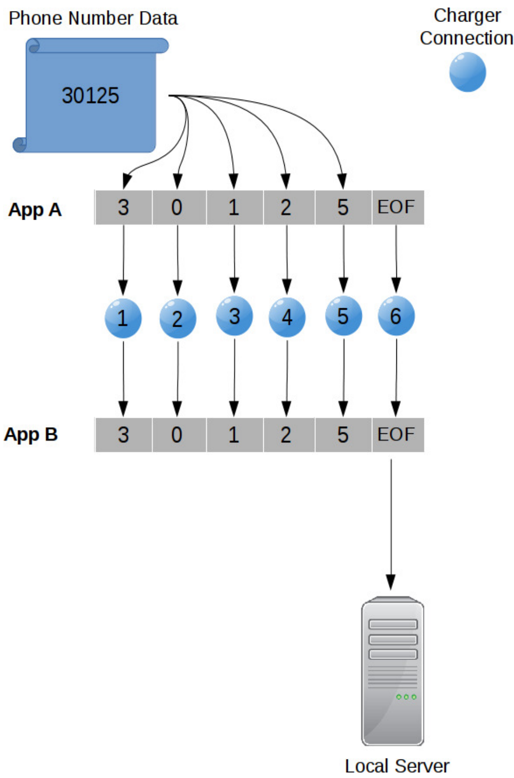


**Figure 4: Chain of covert communication.**

The attack vector commences by the dialer modified app reading the phone number, say "30125". When the smartphone is plugged into the charger, the corresponding event is triggered and the app remains open for 20 - 25ms, which corresponds to the first number "3". This process will be repeated for every number until all numbers are sent in which the final EOF will be sent (by the app remaining open between 0-5ms).

Upon reception of the EOF character, the BBC app realizes that the data transmission is completed, stops monitoring the dialer app and sends the collected data "30125" to our local server. This procedure is illustrated in Fig. 4.

## 4. EVALUATION OF THE SCHEME

As the app installation approval approach involves the proactive participation of the user, we conduct a risk analysis by considering three types of user personas - an ordinary user, a fastidious user and an expert as follows:

- An ordinary user may not detect the malevolent activity, since there are no tangible results on the Android device, which could attract user's attention.

- A fastidious user may use tools like network traffic monitor tools in order to investigate these apps. This user may need to continuously monitor the traffic in order to discover the connection between the BBC modified app and the remote server. If finally this connection is discovered, then the user could eventually find out that the IP belongs to the legitimate company. Thus, no real warning will emerge. Moreover the user will notice no data transmission due to the presence of the CC.

- At this point we consider the case of an expert analyst, who will analyze the two apps. In order to examine the two apps, we consider the cases of static and dynamic analysis.

  1. Static Analysis - This method of testing has distinct advantages in that it can evaluate both web and non-web applications and, through advanced modeling, can detect flaws in the software's inputs and outputs that cannot be seen through dynamic web scanning alone [7]. After the analysis, the Android-manifest files of the two apps will be revealed. These files do not contain any clues for potential connections between the two applications. Thus the examiner needs to investigate each file of the BBC app, in order to find out evidence of the CC. The name of the sending app is concealed inside an obfuscating function, and thus the examiner will not find the name of the app inside the source code. Thus, the examiner should analyze the code in order to find details about the connection between the two apps.
  2. Dynamic Analysis - This type of analysis allows the pair of the apps to run on a system, while their actions are being monitored. When this analysis is preformed, the pair of the two apps will not necessarily cause any concerns to emerge.

The malevolent pair attack proposed in this paper, works on all Android versions, including version 6.0. The method getRunningTasks was deprecated in Android LOLLIPOP (API level 21) [2], because of the possibility of leaking personal information to third party applications. In [11] the method getRunningTasks was used in order to implement a CC attack. This function has merely the same usability for CC attack as the getRunningApps function which is used under no permission need. This function let the malevolent pair to transmit information packets without users' acceptance. We proved this functionality under our proposed attack scheme above. The getRunningApps function has a different usage result on Android 5.1.1 and above, which returns a list of your own application package. In order to implement the same attack on Android 5.1.1 and above versions, we parsed the output of running "ps" command in a shell, which let us get the current running apps.

## 5. DETECTING MALEVOLENT PAIRS

One of the general techniques that was proposed to be applied as countermeasure [14] is Limiting Multitasking. The concept of multitasking allows concurrent execution of tasks which is necessary in applications requiring parallel task execution. However, it enables all the channels that require synchronous communication. This may lead to timing CC.

It is confirmed that two recently proposed architecture modifications and tools that deal with overt and covert channels discovery, TaintDroid [8] and XManDroid [4], still fail to detect several of the implemented channels [13]. Thus, we propose that defences against CC schemes should be implemented inside the Android software. In this way the user should not need to trust third party privacy protection vendors. Solving the confinement problem, and in particular

closing all possible covert channels in a system, is known to be a difficult problem [6, 12]. In [13] it is referred that the mitigation can be achieved at the design time (by reducing access to sensitive APIs or by limiting communication possibilities).

An interesting tool is Poly, an advanced Android application installer, presented in [13]. Poly augments the existing package installer by allowing users to specify their constraints for each permission during installation using a simple and usable interface [13]. However this framework has a substantial impact on the user, who in this case needs to make further trust assumptions and understand the access controls with a great deal of granularity.

Against the above challenges and possible remedies, the proposed solution would need to take into account the following considerations:

1. Retain the usability level. The user should not need to consume more time and effort than before in order to decide on what to install or not.

2. Integrate with the existing Android permission scheme. The new Android permission scheme, should be compatible with the current permission framework.

3. The new scheme should inherently support a much more detailed and fine-grained access to the permissions-related features of each App. The additional information should be accessible to any user that would like to assess the actual privacy and security-related calls of an app. That is, the permissions scheme should admit increased user awareness, without limiting access to APIs or other system functions.

The CC attack techniques exploit functions supported and provided by the Android OS. These functions are system oriented, as they ask for app usage characteristics (CPU, memory, current state, etc). Each CC attack uses one or more of these characteristics in order to succeed. The implementation of such attack performs system queries for obtaining certain app characteristics and eventually setup the CC. Thus, in order to mitigate the CC attacks, each query for app usage characteristics should be declared under certain permissions, warning the user about the potential risks.

The proposed solution extends the Android permission list. This should be done in order to maintain the control over the information that is offered by the system and reduce the risk of not only allowing CCs to manifest but also to attempt to control side and inference channels. The latter can pose threats to privacy. As an example, the battery level is a system information attribute available to all apps without requiring any permissions. In order to illustrate the potential risk, consider an app which interrogates the system every minute to obtain the battery level. This app would effectively have detailed information on how much battery is drained at certain periods. When the app notes increased battery drainage, it can infer that the user is operating the device.

In Android 6.0 the permissions are divided into two separate groups, namely normal and dangerous. Normal permissions pose little risk to the user's privacy or the operation of other apps, while dangerous could potentially affect the user's stored data or the operation of other apps [1]. The battery level is not provided by a dangerous permission, while it can be easily turned to a monitoring tool capturing the user's device usage history. Moreover, the getRunningApps function is also restricted. Yet this is not adequate as the unrestricted information can be obtained by issuing commands that do not require permissions. It is therefore proposed to transform the Android permission scheme from a normal/dangerous classification approach to an architecture enabling the user to perform more informed decisions on their privacy.

We propose dividing the existing permission list into three permission groups, corresponding to system, apps and sensors as follows:

1. The functions revealing information about the system itself, to be grouped under a new permission, named SYSTEM_INFORMATION.

2. The functions revealing information about other apps installed on the system, to be grouped under the permission APP_USAGE_INFORMATION.

3. The functions revealing information from the system sensors, to be grouped under the permission SYSTEM_SENSOR_INFORMATION.

The above grouping of functions will allow the user to highlight the functions relating to a particular app in a more systematic manner.

In this way a novice user will be able to install an app as she normally does in Android 6.0 or earlier, while a more expert user will have the chance to further investigate. This approach does not necessarily prevent any CC, but helps the examiner to identify potential CC attacks. To demonstrate this, consider the proof of concept app pair. When the two apps are installed, the examiner will be informed that one of them (the BBC app), uses the getRunningApps function. As this function provides information about other apps, according to the proposed plan this function will be contained in the APP_USAGE_INFORMATION permission. The examiner will be notified that this permission is used and in the same time she will be informed that the particular function used is getRunningApps.

Moreover, for Android versions 5.1.1 and above the attack does not invoke the getRunningApps function but executes commands that do not require any permissions. It is proposed that the device administrative commands need also to be included in the scheme so that an examiner will have the opportunity to make an informed decision whether to install an app or not.

Furthermore, in [11] a CC attack based on a process priority reporting function was proposed. On a similar note, this function does not ask for any permissions. In our proposed scheme this function could be grouped in the APP_USAGE_INFORMATION permission. Moreover, the user will be informed and alerted that the malevolent app uses the process-priorities function. Another CC enabling function mentioned in the same work [11] is the screen on/off state. This attack in particular uses the isInteractive function through the PowerManager system service, again without the need to request permission. In our proposed scheme this function would be confined by the SYSTEM_INFORMATION permission.

Extending the permission group with these three permissions and organizing the permission scheme in a more informing scheme, we strengthen the defense against CC at-

tacks, protecting users' privacy. Moreover, privacy concerned users will be able to take a first glance about potential risks that an app may pose. Developers who build apps respecting users' privacy may be able to prove their privacy oriented apps, by using as many permissions as they really need, and also as many functionalities of these permissions as they need. In [9] the Stowaway tool was applied to 940 Android applications and it was shown that about one-third of them are over-privileged. The results revealed that applications are in principle over-privileged by only a few permissions, but most additional and unnecessary permissions can be attributed to developer confusion [9].

Consequently, there are two alternatives the app developers seem to follow. One is a scattergun approach, asking for a long list of permissions, but effectively using only a few of them. The other is to be conservative and ask for a limited number of permissions but overuse the enclosed functions. The proposed solution can deal with both approaches as it ties and highlights not only the permissions but also the underlying functions.

# 6. CONCLUSION

The current Android permission schemes provide limited support to a user for making informed decisions relating to their privacy. In addition, the limited scope of the permission schemes exluding a number of system functions, pave the way for CC attacks which effectively circumvent the permissions. In this paper we developed a proof of concept to demonstrate how data exfiltration can be achieved using two apps and two CC.

Two important observations are:

- Permission circumventing attacks like the malevolent pair attack are a real threat for current Android-based smartphones.
- The user awareness about this type of attacks is very low; this fact makes the such attacks even more dangerous.

Interestingly, even though such an attack has already been presented in [13, 14], the user community (including until recently the authors of this work) is still generally not aware of such attacks, and the software vendor has not taken any action since, to effectively encounter this threat.

The proposed work reflects upon the privacy implications of the current permission schemes by proposing a framework for informing the user or examiner on potential privacy leaks and associated risks. As part of future work a formal treatment of the permissions and functions will be further developed and a model with a view to assess the scope, impact and potential of CC in Android devices.

# 7. REFERENCES


[1] http://developer.android.com/guide/topics/security/permissions.html. [Online; accessed 14-February-2016].

[2] http://developer.android.com/reference/android/app/ActivityManager.html. [Online; accessed 14-February-2016].

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

[4] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards taming privilege-escalation attacks on android. In *NDSS*, 2012.

[5] E. Couture. http://www.sans.org/reading-room/whitepapers/detection/covert-channels-33413. [Online; accessed 14-February-2016].

[6] D. E. Denning and P. J. Denning. Data security. *ACM Computing Surveys (CSUR)*, 11(3):227–249, 1979.

[7] N. DuPaul. http://www.veracode.com/products/static-analysis-sast/static-analysis-tool. [Online; accessed 14-February-2016].

[8] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[9] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[10] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96*, pages 104–113. Springer, 1996.

[11] J.-F. Lalande and S. Wendzel. Hiding privacy leaks in android applications using low-attention raising covert channels. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 701–710. IEEE, 2013.

[12] S. B. Lipner. A comment on the confinement problem. In *ACM SIGOPS Operating Systems Review*, volume 9, pages 192–196. ACM, 1975.

[13] C. Marforio, A. Francillon, S. Capkun, S. Capkun, and S. Capkun. *Application collusion attack on the permission-based security model and its implications for modern smartphone systems*. Department of Computer Science, ETH Zürich, Switzerland, 2011.

[14] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun. Analysis of the communication between colluding applications on modern smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 51–60. ACM, 2012.

[15] J. K. Millen. Covert channel capacity. In *null*, page 60. IEEE, 1987.

[16] N. NCSC. Covert channel analysis of trusted systems (light pink book). *NSA/NCSC-Rainbow Series publications*, 1993.

[17] H. Okhravi, S. Bak, and S. T. King. Design, implementation and evaluation of covert channel attacks. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 481–487. IEEE, 2010.

[18] M. Tsavli, P. S. Efraimidis, V. Katos, and L. Mitrou. Reengineering the user: privacy concerns about personal data on smartphones. *Information & Computer Security*, 23(4):394–405, 2015.

[19] J. M. Urban, C. J. Hoofnagle, and S. Li. Mobile phones and privacy. *BCLT Research Paper Series*, 2012.