Move Together and You are Dead: Detecting Local Covert Channels Using Process Activity Correlation on Android Smartphones

ABSTRACT

Modern malware threats utilize many advanced techniques to increase their stealthiness. To this aim, information hiding is becoming one of the preferred approaches, especially to exfiltrate data. However, for the case of smartphones, covert communications are primarily used to bypass the security framework of the device. The most relevant case is when two "colluding applications" cooperate to elude the security policies enforced by the underlying OS. Unfortunately, detecting this type of malware is a challenging task as well as a poorly generalizable process. In this paper, we propose a method for the detection of malware exploiting colluding applications. In more details, we analyze the correlation of processes to spot the unknown pair covertly exchanging information. Experimental results collected on an Android device showcase the effectiveness of the approach, especially when used to detect low-attention raising covert channels, i.e., those active when the user is not operating the smartphone.

1. INTRODUCTION

More than ten years of advancements in security systems imposed malware developers to study new techniques to make their "products" stealthier. To this aim, utilizing some form of information hiding is a relevant trend, which presumably started in 2006 when the Trojan.Downbot [1] allowed to attack and damage numerous institutions during the "Operation Shady RAT". Specifically, the threat hid its presence for months by encoding control commands within HTML pages or JPEG images, which appeared as licit traffic [7].

From 2011, the proliferation of information hiding-capable malware constantly increased and showcased three major techniques to cloak communications: *(i)* methods embedding secrets via digital media steganography, e.g., by manipulating pixels of a picture or samples of an audio track. The most notable examples are Duqu [2] and Trojan.Zbot [13] using digital images as hidden data carriers; *(ii)* methods injecting data within network traffic, e.g., by encoding bits into well-defined values of the throughput. In this class, the most diffuse threats are Regin [15] and Linux.Fokirtor [14] exploiting flows produced by popular network services e.g. the secure shell (SSH); *(iii)* methods hiding information by modulating the status of shared hardware/software resources, e.g., by artificially altering the load of the CPU to signal some events to other processes. A possible example is Airhopper [4] using the GPU of the infected host to produce electromagnetic signals to communicate with the attacker.

As regards targets of such threats, smartphones are generally better suited than desktops as they offer different carriers for embedding secrets. In fact, modern devices natively incorporate camera, GPS, IEEE802.11, Bluetooth, cellular connectivity, and a mixed variety of sensors. However, for the case of smartphones, information hiding is primarily used to implement a local covert channel between two processes running on the same device. This enables to have a pair of "colluding applications" cooperating to void the security framework of the underlying OS [11]. The archetypal approach is Soundcomber [12], which uses information hiding to bypass the security layer of Android devices. In essence, the malware part that has insufficient privileges to access the network implements a covert inter-process communication service with another application having the right permissions to leak data outside the device. Unfortunately, this type of malware is very difficult to spot, since the pair of processes involved in the exchange is unknown, covert channels are characterized by a very low bitrate and often empowered with techniques to reduce their impact on the device (e.g., to avoid performance degradation), and each information hiding technique is tightly coupled with the carrier, thus making the detection poorly generalizable.

In this perspective, we propose a method to detect the presence of a local covert channel for mobile devices based on the activity statistics of processes. In more details, we identify the pairs of processes active at the same time, and we use an ad-hoc metric to recognize whether they are covertly exchanging data. In order to avoid a loss of generality, we investigate when the phone is idle (i.e., the user is inactive) and when it is busy (i.e., the user interacts with the smartphone, for instance to watch videos or browse the Web).

The main contribution of the paper is the definition of a framework to detect malware using a colluding applications

SAC'16, April 4-8, 2016, Pisa, Italy

Copyright 2016 ACM 978-1-4503-3739-7/16/04...$15.00

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

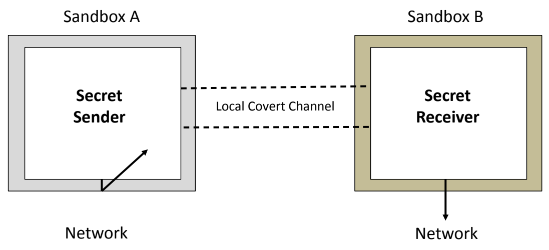


Figure 1: Colluding applications scenario: `Secret Sender` and `Secret Receiver` using some form of information hiding to establish a local covert channels to exchange data outside their sandboxes.

architecture. Nevertheless, since we rely upon high-level indicators, the proposed approach is general enough to be applied to different information hiding-capable malware.

The rest of the paper is structured as follows. Section 2 provides the background and surveys works dealing with the colluding applications threat. Section 3 presents the proposed detection method, while Section 4 deals with the experimental results collected on a real Android device in different use cases. Lastly, Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

The most notable example of information hiding-capable mobile malware is Soundcomber [12], which covertly transmits the keys pressed during a call, e.g., to exfiltrate a PIN of a bank service. Since the sensitive data is acquired by using the microphone, some security/privacy policies of the OS could prevent the malware to access the network. To bypass such a limit, the part that has insufficient privileges can use a colluding application to leak data outside the device. The latter could be an innocent looking software, for instance an organizer or a game, having the access to the protocol stack. Figure 1 depicts such a use case, which is commonly defined as the "colluding applications" scenario [10]. In more details, the `Secret Sender` has access to the data but has not the permission to access the network, while the `Secret Receiver` has access to the network and will be able to exfiltrate the received data to an external server. To avoid its detection, the produced traffic can be further injected into another covert channel or concealed through obfuscation/encryption techniques such as Tor (see, e.g., Simplocker [9]). Therefore, this also motivates the need of early recognizing the presence of the malware when it still operates within the device.

To implement a local covert channel between the colluding applications, Soundcomber exploits the most popular functionalities of modern smartphones and mobile devices. Specifically: the vibration or volume settings (one process is differencing the status of the vibration/volume and the other infers secret data bits from this event), the screen state (secret bits are transferred by acquiring and releasing the wake-lock permission that controls the screen state), and file locks (secret bits are exchanged between the processes by competing for the exclusive access to a file).

In general, covert channels can work synchronously or asynchronously. When colluding processes operate synchronously, they must be both simultaneously active when communicating. Therefore, as soon as the resource used by the `Secret Sender` is modified (e.g., the volume of the ringtone is changed), this must be instantly detected and analyzed by the `Secret Receiver`. Conversely, when colluding applications operate asynchronously, modifications made by the `Secret Sender` can be acquired by the `Secret Receiver` at any time. As a consequence, processes could not be active at the same time. In this paper we focus on synchronous channels since they are more efficient and widely adopted in malware [11].

Along the lines of Soundcomber, other techniques for covert communication on a smartphone have been recently proposed. For instance, the work by Marforio et al. [10] investigates seven new methods using different locks simultaneously as to improve the throughput of the covert channel. Later, Lalande and Wendzel [6] described additional ways to enable local covert channels for mobile devices. While designing the channels, authors primarily focused on how to reduce their "attention raising" characteristics. Especially, they propose to activate the malware when the user is not performing any task with his/her smartphone (denoted in the following as idle state), since establishing the covert channel during user's activity could degrade the smartphone performances and reveal the presence of the ongoing attack.

As regards detection of local covert channels, the literature presents only a very limited amount of work, thus highlighting difficulties in developing effective or general countermeasures. The most relevant work is [5], where Hansen et al. try to detect three types of covert channels (i.e., those based upon vibration, volume, and wake lock). Specifically, authors propose to count the number of events, which are the changes of settings related to vibration, volume and the wake lock, and evaluate them into time windows to detect anomalous bursts. Unfortunately, the decision is made by comparing the number of events against a threshold: it requires to know the type of events and to fix the threshold in advance. Thus, in this paper, we intend to propose a more general detection technique that overcomes such a limitation.

From the general Android security perspective, literature mainly showcases efforts to track information flows [3], which is a mandatory preliminary step before considering local covert channels. Another relevant field deals with the extension of existing solutions such as Taintdroid [3] or inter component analysis approaches [8]. However, they adoption to detect a violation of the information policy by local covert channels is still an open research problem.

3. DETECTION METHODOLOGY

In this section, we describe the detection method used to identify the unknown pair of processes acting as a colluding applications threat. To mitigate the complexity and guarantee an efficient implementation on many devices, we only use activity statistics provided by the Android OS and we introduce a condensed metric.

In more details, for each process, we gather from `/proc/[pid]/stat` the following values: *i*) `utime` – the amount of time that the process has been scheduled in user mode, *ii*) `stime` – the amount of time that the process has been scheduled in kernel mode, *iii*) `cutime` – the amount of time that the process waited for children when scheduled in user mode, and *iv*) `cstime` – the amount of time that the process waited for children when scheduled in kernel mode. Then, the number of ticks assigned to the process from its creation is given by the sum of all values, i.e., `utime` + `stime` + `cutime` + `cstime`. The measure is repeated after a Δt, which is randomly selected from a uniform distribution. This allows to reduce the impact of the proposed framework over the device in terms of battery drains and CPU usage.

For each process, the number of cycles allocated between two adjacent samplings are evaluated by computing their difference, and we define such a quantity as M_i. For example, if the measurement of a process at the instant i is equal to 713 clock cycles, and the value changes to 715 at $i+1$, the difference is of 2 clock ticks, thus $M_i = 2$. This means that the process has performed some operations during the period of time between two measurements, i.e., Δt. However it must be noted that such a mechanism could fail to correctly identify an active process, especially when in presence of a very limited number of performed operations.

Therefore, to decide whether a process is active, we introduce the following three alternative *decision rules*:

1. *threshold*: the process is considered active if the number of ticks M_i exceeds a given threshold T.
2. *relative*: the process is considered active if the number of ticks M_i is greater than the average number of ticks consumed by the other processes running on the device.
3. *sliding window*: the process is considered active if the average of the last N measurements (i.e., M_{i-N-1}, M_{i-N-2}, $\ldots$, M_{i-1}) is lower than the current measured value M_i.

Upon choosing a variant of the decision rule, a process can be considered active/inactive. However, this gives only "instantaneous" information. Thus, we introduce a total counter, denoted as $T_C(x)$ considering the time for which the given process x is active during its lifetime. Table 1 shows an example of results for $T_C(x)$, for three processes a, b and c during five consecutive measurements.

As a second step, we can evaluate the behavior of all the pairs of processes running on the smartphone in order to recognize if they are two colluding applications. To this aim, we introduce the activity counter denoted as $A_C(x,y)$ representing the number of measurements for which processes x and y were both active.

The two metrics defined above are then used to obtain a condensed indicator, defined as Activity Factor (A_F), to decide whether a pair of processes is covertly exchanging data. Specifically:

$$A_F(x,y) = \frac{A_C(x,y)}{T_C(x) + T_C(y) - A_C(x,y)} \times 100 \qquad (1)$$

where, $A_C(x,y)$ is the activity counter of the pair x and y, $T_C(x)$ and $T_C(y)$ the total counter for process x and y, respectively. In more details, $A_F(x,y)$ allows to assess the correlation between processes by offering a percentage of "concurrent" activity. Hence, it can be used as a condensed indicator to judge whether a pair of processes is implementing a colluding applications threat and exchanging data through a local covert channel. Its simple nature allows an easy implementation on resource-limited devices and reduces impacts in terms of performance degradation if used in real-time. As an example, Table 2 reports computed $A_F(x,y)$ values for the scenario previously presented in Table 1. Obviously, the proposed approach must be properly tuned. Section 4 discusses how to experimentally find the most suitable values for the A_F to detect colluding applications both when the phone is idle and when the user performs some operations.

Table 1: Exemplary calculation of the total counter $T_C(x)$ for processes a, b and c during five consecutive measurements.

Measure	Process a	Process b	Process c
M_1	Active	Active	Not active
M_2	Not active	Active	Not active
M_3	Active	Active	Active
M_4	Not active	Active	Active
M_5	Active	Active	Active
$T_C(x)$	$T_C(a) = 3$	$T_C(b) = 5$	$T_C(c) = 3$

Table 2: Exemplary calculation of the activity factor $A_F(x,y)$ for processes a, b and c.

Pair	Activity	$A_F(x,y)$ [%]
a, b	Active during 1, 2, 3	$A_F(a,b) = \frac{3}{(3+5-3)} \cdot 100 = 60$
a, c	Active during 3, 5	$A_F(a,c) = \frac{2}{(3+3-2)} \cdot 100 = 50$
b, c	Active during 3, 4, 5	$A_F(b,c) = \frac{3}{(3+5-3)} \cdot 100 = 60$

4. EXPERIMENTS AND RESULTS

In order to have a realistic testbed, we implemented the following three state-of-the-art information hiding techniques used to establish a local covert channel between the colluding applications on the Android OS-based device. Specifically:

- *File Lock* [12]: the `Secret Sender` sends a '1' by locking the file. The `Secret Receiver` also tries to acquire the same lock, and if it succeeds, the action is interpreted as a '0'. On the contrary, if the `Secret Sender` has locked the file before the `Secret Receiver`, an exception is raised and a '1' is inferred.
- *Volume Settings* [10,12]: the secret data is encoded in the volume level of the smartphone. Then, its variation can be used by the `Secret Sender` to transmit information towards the `Secret Receiver`.
- *Type of Intent* [10]: in Android OS, an intent is a kind of message among a sender and a set of registered listeners. The `Secret Receiver` subscribes to one or more intents and the `Secret Sender` communicates by encoding data in the selected intent. Since Android offers 256 types of intents, each one can be mapped into a different byte value.

We point out that with the type of intent channel it is possible to transmit 1 byte between the `Secret Sender` and

`Secret Receiver` at once, with the volume settings 4 bits (when an application using a 16-valued audio output is used), and only 1 bit with the file lock method. Such behaviors impact over the throughput as well as the duration of the transmission, and possibly, on the detectability of the colluding applications.

For each method, we used the colluding applications to exchange a message of 100 bytes as to simulate the exfiltration of sensitive information, e.g. an entry of the address book (name, email, phone number). Each trial lasted 20 minutes and has been repeated 10 times as to have average values with a proper confidence level. As regards the detection method, the time between two consecutive measurements Δt was randomly chosen from a uniform distribution ranging from 200ms to 2s. All the trials have been repeated in two scenarios: when the smartphone is in idle and when the user is performing some activity.

4.1 Scenario 1: idle phone

In this section, we consider the smartphone as idle, i.e., there is no deliberate user's activity.

The first results are collected by using the decision rule based on a threshold T presented in Section 3. Table 3 showcases the values for $A_C(x, y)$, $T_C(x) + T_C(y)$ and $A_F(x, y)$ for the pair of processes involved in the covert channel communication. In the following, to avoid burdening the notation, we drop the dependence from the process, i.e., we simply refer to A_C, T_C and A_F.

Since the parameter T plays a major role, we investigated different values, i.e., $T = 1, 2, 5, 10$. For all the covert channels we can observe that if T is equal to 1 or 2 clock ticks, the standard deviation σ of A_F has the best values, thus reducing statistical errors. On the contrary, as T increases "slow" processes could remain undetected and marked as idle for long bursts of measurements. Therefore, using such a condensed parameter to discriminate if a pair of processes are colluding could lead to incoherent results or false positives/negatives due to the high σ.

Figure 2 portraits the A_F for the five most active pairs of processes, for $T = 1, 2, 5, 10$, and when data is exchanged through the type of intent covert channel. In this case, the A_F value of the pair of colluding applications (labeled as 'A' in the figure) is greater than the one of other processes, thus allowing to easily recognize them. As it can be seen, the lower the value of T, the better is the "spread" of A_F among the most interacting processes, thus making the detection of colluding applications more clear and accurate. Figure 3 and 4 present the same analysis for the two other types of covert channels, the volume settings and the file lock, respectively. Results are very similar to those obtained for the type of intent channel, and the best activity factor is obtained for smaller values of T.

Then, we present results when using the "relative" decision rule, i.e., the pair of processes is active when it gets more clock ticks than the average of other entities running on the device. Table 5 presents the values for the parameters of the processes acting as colluding applications. The best performances are obtained when detecting colluding applications exchanging data through the file lock method. This is due to the behavior of the channel, which is active for a longer period if compared to other methods. In fact, when using the file lock, the data transmission lasts 8 times longer than for the type of intent case. Figure 8 showcases the five most active pair of processes sorted according to the A_F, and again, our method correctly identifies the pair 'A' as the colluding one. As it can be noted, better results are obtained for hiding methods that send secret data in smaller parts and thus require more time to complete the transfer. Moreover, for the covert channel using the file lock information hiding scheme, the highest A_F also has the lower σ compared to the other two channels. Hence, such method can be recognized with a greater accuracy.

Lastly, we investigate the results in detecting the pairs of applications covertly exchanging data when the sliding window rule is adopted. Table 4 presents obtained experimental results for all three considered covert channels, and the best performances are achieved when detecting the colluding applications sending information through the file lock covert channel. Figures 5–7 illustrate the A_F results for the five most active process pairs for this detection method, and also in this case, 'A' is the couple implementing the colluding applications threat. However, despite the high value of σ, colluding applications can be clearly identified, and also in this case, the detection of threats using the file lock covert channel is the most accurate.

To summarize, all the decision rules used to mark a process as active allowed to use our method based on the condensed A_F metric to identify colluding applications. In more details, the variant using the threshold T yielded to the best results, and the file lock covert channel was the one that can be detected with the best accuracy. It must be also noted that in majority of cases a pair of processes having an $A_F > 60\%$ can be considered suspicious as the other pairs of processes typically did not reach higher than 10%. Therefore, as a preliminary step, we do consider colluding applications those having an $A_F > 60\%$. Obviously, this aspect requires also a proper modeling and we consider it as a part of our ongoing research. However, when in presence of high values of σ false positives/negatives could require further investigations, for instance considering also CPU/memory usage statistics.

4.2 Scenario 2: user activity

As said, the best performances in terms of spotted applications are achieved when active processes are identified with the threshold method and $T = 1$. Therefore, for the case of a busy smartphone, we limit our considerations to such a case. Besides, for the sake of brevity, we limit our investigation to the file lock covert channel, since it is characterized by a very limited bandwidth and more difficult to recognize, at least from a theoretical point of view. The duration of this round of tests is 15 minutes, and the pattern characterizing the user is subdivided into 5 minutes slots as follows: download of a video stream, Facebook and writing, and web browsing.

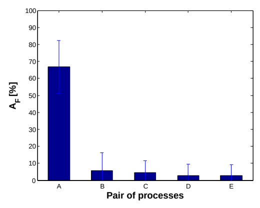
Concerning the parameters used in our framework, the pair of processes implementing the colluding applications threat are characterized by, on the average, $A_C = 54.50$ and $\sigma = 2.12$, $T_C = 94.80$ and $\sigma = 6.16$, and $A_F = 57.65$ and

Table 3: Statistics for the pair of processes identified as colluding applications when using the threshold decision rule.

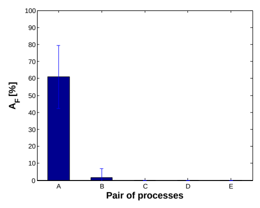
T	Activity counter A_C		Total counter T_C		Activity Factor A_F [%]	
	Avg.	σ	Avg.	σ	Avg.	σ
Type of Intent covert channel						
1	8.20	9.14	14.50	21.53	66.80	15.41
2	2.70	1.25	4.40	1.58	60.83	18.45
5	2.10	1.60	7.10	9.85	55.40	32.65
10	1.00	0.94	2.60	4.74	56.88	47.73
Volume Settings covert channel						
1	11.60	8.57	16.60	14.32	72.97	11.09
2	3.40	1.17	5.20	1.55	64.71	11.10
5	6.10	4.04	13.10	10.30	58.04	22.15
10	0.70	0.48	1.30	0.48	55.00	43.78
File Lock covert channel						
1	70.70	3.50	76.70	4.35	92.27	3.42
2	34.50	3.44	37.70	4.16	91.76	5.54
5	25.90	19.23	32.40	22.86	77.07	9.80
10	1.00	0.47	1.80	0.79	60.00	32.58

Table 4: Statistics for the pair of processes identified as colluding applications when using the sliding window decision rule.

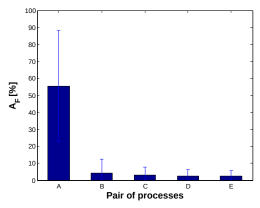
N	Activity counter A_C		Total counter T_C		Activity Factor A_F [%]	
	Avg.	σ	Avg.	σ	Avg.	σ
Type of Intent covert channel						
2	7.30	5.96	14.00	17.11	64.73	19.98
5	4.50	1.27	5.50	2.95	89.26	18.82
10	4.20	1.87	9.70	11.14	73.56	34.60
20	5.60	5.36	9.60	9.11	61.01	16.91
Volume Settings covert channel						
2	9.50	2.68	18.90	15.67	67.94	28.92
5	7.60	2.32	9.70	2.50	79.01	13.69
10	12.80	6.23	21.10	10.41	63.83	15.70
20	4.60	0.97	7.50	2.01	62.83	8.84
File Lock covert channel						
2	33.00	4.35	39.40	6.36	84.19	4.48
5	33.10	3.41	43.50	5.84	76.56	6.20
10	58.40	19.90	91.50	34.79	65.79	10.98
20	31.00	5.12	38.10	4.51	81.17	7.61



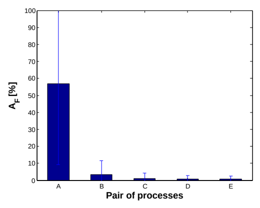
(a) T = 1



(b) T = 2

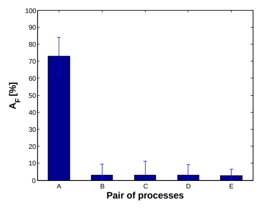


(c) T = 5

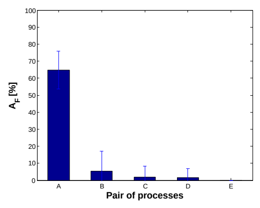


(d) T = 10

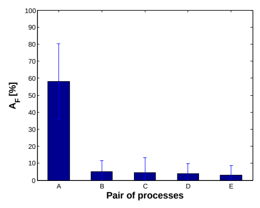
Figure 2: Five most active pairs of processes when measuring activity using the threshold decision rule for the type of intent covert channel.



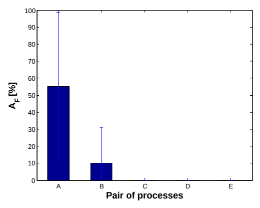
(a) T = 1



(b) T = 2

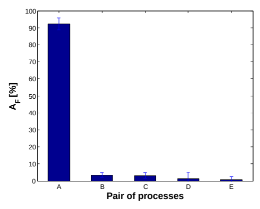


(c) T = 5

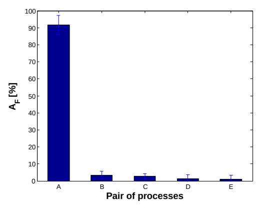


(d) T = 10

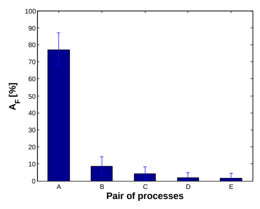
Figure 3: Five most active pairs of processes when measuring activity using the threshold decision rule for the volume settings covert channel.



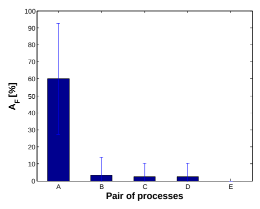
(a) T = 1



(b) T = 2

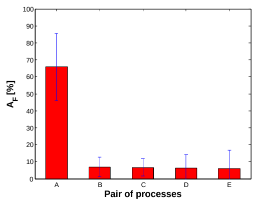


(c) T = 5

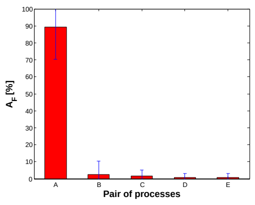


(d) T = 10

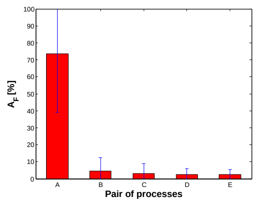
Figure 4: Five most active pairs of processes when measuring activity using the threshold decision rule for the file lock covert channel.



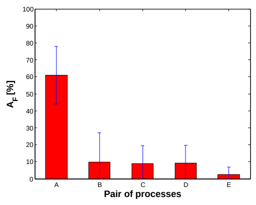
(a) N = 2



(b) N = 5

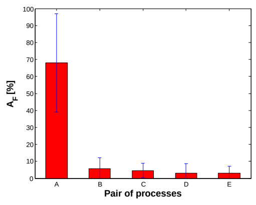


(c) N = 10

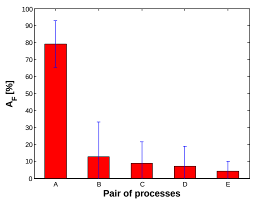


(d) N = 20

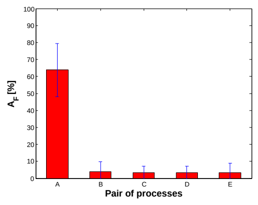
Figure 5: Five most active pairs of processes when measuring activity using the sliding window decision rule for the type of intent covert channel.



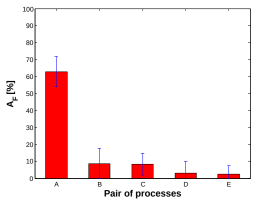
(a) N = 2



(b) N = 5

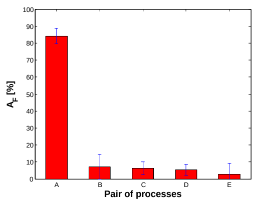


(c) N = 10

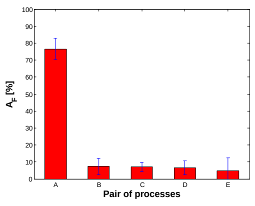


(d) N = 20

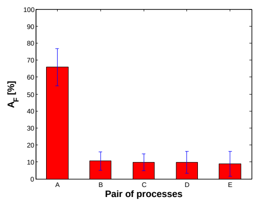
Figure 6: Five most active pairs of processes when measuring activity using the sliding window decision rule for the volume settings covert channel.



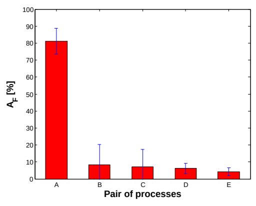
(a) N = 2



(b) N = 5



(c) N = 10

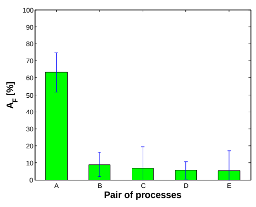


(d) N = 20

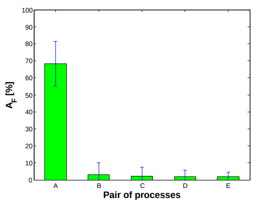
Figure 7: Five most active pairs of processes when measuring activity using the sliding window decision rule for the file lock covert channel.

Table 5: Statistics for the colluding applications using the relative decision rule.

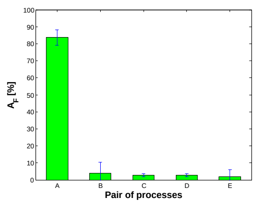
Covert Channel	A_C		T_C		A_F [%]	
Type	Avg.	σ	Avg.	σ	Avg.	σ
Intent	12.30	8.14	19.90	13.20	63.22	11.51
Volume	23.50	6.00	37.60	20.08	68.31	13.06
File Lock	95.70	11.00	114.60	13.76	83.68	4.48



(a) Type of intent covert channel



(b) Volume settings covert channel



(c) File lock covert channel

Figure 8: Five most active pairs of processes when measuring activity using the relative decision rule.

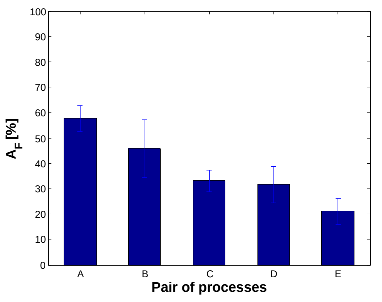


Figure 9: Five most active process pairs when activity is measured with the threshold rule and $T = 1$ for the file lock covert channel.

$\sigma = 5.02$. Besides, Figure 9 depicts the five processes having the highest concurrent activity, and also in this case, 'A' denotes the pair of colluding applications. As shown, even if the user is interacting with the device, the A_F of the pair exploiting information hiding is the highest one, but its average value is lower than in the idle case. Moreover, pairs B–E achieve significantly higher A_F compared to when the smartphone is in idle, e.g., B has $A_F > 40\%$. By analyzing the most correlated processes, it turns out that they are tightly coupled with the behavior of the user. For instance, pairs B and C aggressively interact due to the presence of the *com.touchtype.swiftkey* process, i.e., an on-screen keyboard used by the Facebook App and the Web Browser, respectively. Thus, also in this scenario, colluding applications can be spotted by simply evaluating the A_F, yet its variance would lead to many false positives. Therefore, when in presence of a busy device, the proposed approach should be supported by a minimal knowledge of the processes used by the Android OS to offer system-wide services. In other words, the "black box" approach used for the idle case (i.e., is not relevant the purpose of processes) could partially reduce its performances when in presence of applications tightly interacting with other software components. Luckily, having a sort of database of well-known "clean" applications supposed to offer support to other components is straightforward.

5. CONCLUSION

In this paper we presented a method to detect local covert channels on Android-based mobile devices. In essence, it relies upon a condensed metric defined as activity factor used to spot the pair of processes implementing a colluding application threat. Since the decision rule used to mark a process as active played a role, we investigated three different variants by means of lab trials on an Android smartphone.

Experimental results showcased the feasibility and effectiveness of the proposed approach, especially to detect malware using low-attention raising local covert channels, i.e., those that perform their activity when the user is idle. As mentioned, this is the preferred way of functioning for mobile malware since establishing the covert channel during user's activity could degrade the device performance and reveal the presence of the ongoing infection. Therefore, information hiding-capable malware developers are now facing a dilemma: both initiating malicious software while the user is active or during phone idle state could reveal its activities – the only reasonable way to still remain undetected is to intentionally lower the covert transmission rates. In result, this makes such a data exfiltration method less effective.

Future works aims at enrich the usage-pattern of a busy phone as to have a more comprehensive set of use cases. Besides, developing more sophisticated decision rules based on the processes' activities as well as transforming the proposed framework to a production-quality Android tool are parts of our ongoing research.

6. REFERENCES

[1] Éamonn Young and E. Ward. Trojan.downbot, May 2011. https://www.symantec.com/security_response/writeup.jsp?docid=2011-052413-1248-99.

[2] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. Duqu: A stuxnet-like malware found in the wild. Technical report, CrySyS Lab Technical Report, Apr. 2014.

[3] W. Enck, P. Gilbert, 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. In *9th USENIX Symposium on Operating Systems Design and Implementation*, pages 393–407, Vancouver, BC, Canada, Oct. 2010. USENIX Association.

[4] M. Guri, G. Kedma, A. Kachlon, and Y. Elovici. AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies. In *2014 9th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 58–67, The Americas, Oct. 2014. IEEE.

[5] M. Hansen, R. Hill, and S. Wimberly. Detecting covert communication on Android. In *37th Annual IEEE Conference on Local Computer Networks*, pages 300–303, Clearwater, Florida, USA, Oct. 2012. IEEE Computer Society.

[6] J.-F. Lalande and S. Wendzel. Hiding privacy leaks in Android applications using low-attention raising covert channels. In *First International Workshop on Emerging Cyberthreats and Countermeasures*, pages 701–710, Regensburg, Germany, Sept. 2013. IEEE Computer Society.

[7] H. Lau. The truth behind the shady RAT, 2011. http://www.symantec.com/connect/blogs/truth-behind-shady-rat.

[8] L. Li, A. Bartel, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, P. Mcdaniel, Y. Le Traon, S. Arzt, R. Siegfried, E. Bodden, D. Octeau, and P. Mcdaniel. I know what leaked in your pocket: uncovering privacy leaks on Android Apps with Static Taint Analysis. Technical report, Apr. 2014.

[9] R. Lipovsky. Eset analyzes simplocker – first android file-encrypting, tor-enabled ransomware, June 2014. http://www.welivesecurity.com/2014/06/04/simplocker.

[10] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun. Analysis of the communication between

colluding applications on modern smartphones. In *28th Annual Computer Security Applications Conference*, pages 51–60, Orlando, Florida, USA, Dec. 2012. ACM Press.

[11] W. Mazurczyk and L. Caviglione. Steganography in Modern Smartphones and Mitigation Techniques. *IEEE Communications Surveys & Tutorials*, 17(1):334–357, Jan. 2015.

[12] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Network and Distributed System Security Symposium*, San Diego, California, USA, Feb. 2011. The Internet Society.

[13] J. Segura. Hiding in plain sight: a story about a sneaky banking trojan, Feb. 2014. https://blog.malwarebytes.org/security-threat/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/.

[14] Symantec Security Response. Linux back door uses covert communication protocol, 2013. http://www.symantec.com/connect/blogs/linux-back-door-uses-covert-communication-protocol.

[15] Symantec Security Response. Regin: Top-tier espionage tool enables stealthy surveillance, 2014. http://www.symantec.com/connect/blogs/regin-top-tier-espionage-tool-enables-stealthy-surveillance.