A-SIT Technology
E-Government Innovationszentrum

25. Oktober 2021 Studie · Mobile Sicherheit · Privatsphäre & Krypto · 15 Min. Lesezeit

Cross-User Sicherheit unter Android

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Stand 2021-10-15
Quelle https://technology2.a-sit.at/veroeffentlichungen/cross-user-sicherheit-unter-android/

In diesem Bericht wird die Sicherheit der Isolation zwischen verschiedenen Benutzerkonten unter Android genauer beleuchtet. Insbesondere wird dabei das sogenannte Arbeitsprofil genauer unter die Lupe genommen. Dieses soll es ermöglichen, Arbeitsdaten sicher auf dem auch privat genutzten Smartphone zu speichern. Dazu wird vorgestellt, wie diese Funktionalität unter Android technisch umgesetzt wurde. In Experimenten wird weiters untersucht, ob die Isolation der geschäftlichen Daten umgangen werden kann. Dabei wird konkret untersucht, ob eine Applikation aus dem privaten Bereich Rückschlüsse auf Aktionen im Arbeitsbereich ziehen kann. Im Experiment wird gezeigt, dass es möglich ist, durch die Analyse von der Applikation zugänglichen Daten Rückschlüsse auf im Arbeitsbereich geöffnete Webseiten zu ziehen.

1. Einleitung

Smartphones sind für viele Nutzerinnen und Nutzer zum meistverwendeten elektronischen Gerät geworden. Dadurch sammeln sich auf diesen Geräten viele persönliche Daten wie Fotos und Nachrichten. Um nicht mehrere Geräte gleichzeitig herumtragen zu müssen und Kosten zu sparen, wird das private Gerät auch oft für die Arbeit verwendet („bring your own device" – BYOD). Werden diese Geräte also sowohl für private als auch geschäftliche Zwecke verwendet, so befindet sich ein Datenschatz auf den Geräten, der aus unterschiedlichen Gründen und vor unterschiedlichen Akteuren geschützt werden muss. Gegen den Zugriff durch lokale Angreifer (beispielsweise nach Verlust) wird in der Regel ein wissensbasierter oder biometrischer Faktor wie PIN oder Fingerabdruck verwendet. Doch auch die Daten am Smartphone selbst sollten nicht jeder Applikation zugänglich sein. So sollte eine private Applikation oder ein heruntergeladenes Spiel keinen Zugriff auf Arbeitsdaten haben. Ebenso benötigen Applikationen aus dem Arbeitsbereich nicht unbedingt uneingeschränkten Zugriff auf die persönlichen Daten welche am Gerät gespeichert © A-SIT, Partner für Sicherheitsfragen sind. Um dies umzusetzen, verwendet Android seine Multinutzerfunktionalität, welche wiederum auf der Multinutzerfunktionalität des verwendeten Linux-Unterbaus aufbaut. Wie diese unter Android verwendet ist beleuchtet das nächste Kapitel genauer. In diesem Bericht wird weiters untersucht, ob durch private Applikationen eine Gefahr für die im Arbeitsbereich gespeicherten Daten sowie anderen Informationen besteht.

2. Hintergrund

Android unterstützt mehrere Benutzerkonten auf einem Gerät. Dabei werden eingerichtete Konten, nutzerspezifische Applikationsdaten und weitere auf dem Gerät gespeicherte Benutzerdaten strikt separiert. [1] Dadurch kann ein Gerät wie zum Beispiel ein Tablet von mehreren Nutzerinnen und Nutzern verwendet werden. Hierbei wird zwischen drei verschiedenen Benutzertypen unterschieden. Der primäre Benutzer hat Zugriff auf alle Einstellungsmöglichkeiten und kann nicht entfernt werden. Dieser ist im Hintergrund immer aktiv. Bei den untergeordneten (secondary) Benutzern handelt es sich um alle Benutzer die dem Gerät die neben dem immer vorhandenen primären Benutzer hinzugefügt wurden. Diese können jederzeit entfernt werden und können keinen Einfluss auf andere Benutzer am Gerät ausüben. Diese Benutzer können im Hintergrund weiterlaufen oder temporär deaktiviert werden. Weiters gibt es auch einen Gastnutzer, der für die temporäre Benutzung des Geräts bestimmt ist.

Aus Administrationssicht gibt es zwei verschiedene Profiltypen. Zum einen gibt es die beschränkten (restricted) Profile. Dabei handelt es sich um Benutzer welche dem primären Benutzer unterstehen, also untergeordnete Benutzer und Gastnutzer wie im vorherigen Absatz beschrieben. Weiters gibt es aber auch noch verwaltete (managed) Profile. Diese können von einem Administrator gesteuert werden, sind jedoch aus Benutzersicht mit dem primären Profil verschmolzen. Dadurch ist es möglich, private und geschäftliche Applikationen gleichzeitig auszuführen. Diese Funktionalität wird aus Benutzersicht auch als Arbeitsprofil bezeichnet. Mit diesen Arbeitsprofilen gibt es also eine Möglichkeit, um Arbeitsdaten von persönlichen Daten und Applikationen zu trennen. Diese Funktion basiert auch auf Androids Mehrbenutzerfunktionalität und hat zum Ziel sowohl die Arbeitsdaten als auch die Privatsphäre der Benutzerin bzw. des Benutzers zu schützen [2]. Im Vergleich zu beschränkten Profilen muss aus Sicht der Benutzerin bzw. des Benutzers nicht zwischen den verschiedenen Benutzerprofilen gewechselt werden. Die privaten und geschäftlichen Applikationen teilen sich also einen Applikationsstarter, den Benachrichtigungsbereich und den Applikationswechsler. Der Administrator des Unternehmens hat – mit wenigen Ausnahmen wie beispielsweise der Bildschirmsperre – nur Zugriff auf Einstellungen die den Arbeitsbereich betreffen. Applikationen aus dem Arbeitsbereich werden zur Unterscheidung von persönlichen Applikationen mit einer Brieftasche dargestellt. Applikationen aus den beiden Bereichen können über die folgenden limitierten Kanäle miteinander kommunizieren bzw. auf Daten des jeweils anderen Bereiches zugreifen [3]:

- Applikationen aus beiden Bereichen können Intents starten. Dies erlaubt es, andere Applikationen zu starten sowie Daten zu übergeben. Die Benutzerin bzw. der Benutzer hat dabei in der Regel die Möglichkeit auszuwählen von welcher Applikation ein Intent behandelt wird. Ist ein Arbeitsprofil aktiviert, so können auch Applikationen aus dem Arbeitsbereich gewählt werden, sofern diese Funktion nicht durch den Administrator deaktiviert wurde.

- Wenn größere Dateien zwischen den Profilen ausgetauscht werden, so muss ein sogenannter Content Provider verwendet werden. Normale Dateipfade können im jeweils anderen Profil nicht aufgelöst werden, da die beiden Profile einen eigenen Speicherbereich haben und so nicht direkt auf Daten des jeweils anderen Profils zugreifen können.

- Da sich sowohl Arbeitsprofil als auch Privatbereich den Benachrichtigungsbereich teilen ist der Zugriff auf Benachrichtigungen beschränkt. Applikationen im Arbeitsbereich können zwar Benachrichtigungen senden, jedoch nicht auf andere Benachrichtigungen im

Benachrichtigungsbereich zugreifen. Applikationen im Privatbereich können jedoch auf Benachrichtigungen, welche im Arbeitsbereich erstellt wurden, zugreifen. Dies jedoch nur, wenn dies vom Administrator des Arbeitsprofils erlaubt wurde.

All diese Kanäle können jedoch durch einen Administrator über den DevicePolicyManager weiter beschränkt oder komplett deaktiviert werden [4].

3. Verwandte Arbeiten

Die Isolation zwischen den verschiedenen Profilen und damit der Schutz der gespeicherten Daten beruht auf verschiedenen Funktionen des Betriebssystems bzw. des verwendeten Linux-Kernels. Gibt es in einem der verwendeten Komponenten eine Sicherheitslücke, so kann durch Ausnutzung dieser die Isolation potentiell umgangen werden. In [5] werden verschiedene bekannte Klassen von Sicherheitslücken auf ihren Einfluss auf die Isolation der Multibenutzerfunktionalität von Android betrachtet. Um den Schutz gegen bekannte Schwachstellen aufrecht zu erhalten sollten also Geräte mit aktuellen Sicherheitsaktualisierungen eingesetzt werden. Sind solche Aktualisierungen verfügbar, so können diese auch vom Administrator eingespielt werden [6].

In [7] werden mögliche Angriffsszenarien auf das Arbeitsprofil beleuchtet. Hierbei werden im Wesentlichen zwei mögliche Wege beleuchtet. Zum einen kann die Separierung durch Erlangen von Root-Rechten umgangen werden. Durch Erlangen von Root-Rechten hat ein Angreifer vollen Zugriff auf die im Arbeitsprofil gespeicherten Daten, sobald dieses entsperrt wurde. Ebenso kann durch einen Angriff auf den Binder IPC Service auf Daten im Arbeitsprofil zugegriffen werden. Der Binder IPC Service [8] ist eine wesentliche Komponente unter Android um eine Kommunikation zwischen verschiedenen Prozessen zu ermöglichen. Da diese Komponente auch von Systemdiensten wie dem Fenstermanager oder den Eingabemethoden verwendet wird, wäre eine Schwachstelle im Binder verheerend für die Sicherheit des Arbeitsprofils. So könnte ein Keylogger alle im Arbeitsprofil getätigten Eingaben mitschneiden. Dadurch zeigt auch diese Arbeit, dass es wichtig ist, dass die verwendeten Geräte aktualisiert sind und somit keine bekannten (Binder-)Schwachstellen haben und dass die Geräte für Modifikationen gesperrt sind um Änderungen an den Systemdiensten wie dem Binder zu verhindern.

4. Methodik

Da bei Verwendung des Arbeitsprofils sowohl private als auch geschäftliche Applikationen auf demselben Gerät ausgeführt werden können sind sowohl aktive als auch passive Angriffe möglich. Unter aktive Angriffe klassifizieren wir in diesem Bericht Angriffe, bei denen die angreifende Applikation zeitgleich mit der zu erkennenden Aktion ausgeführt werden muss. Bei den passiven Angriffen kann die Applikation jederzeit versuchen Daten auszulesen.

Bei den passiven Angriffsmöglichkeiten konzentrieren wir uns in den Experimenten auf Timing-Angriffe auf die Schnittstellen zwischen dem privaten Profil und dem Arbeitsprofil. Im Einzelnen wird versucht, die Existenz von Applikationen als auch Dateien im Arbeitsprofil zu erkennen. Um Applikationen zu erkennen wird versucht beispielsweise Intents des Arbeitsprofils zu starten oder Methoden des PackageManagers mit Parametern von im Arbeitsprofil vorhandenen Applikationen auszuführen. Um die Existenz von Dateien zu erkennen, wird versucht auf vorhandene Dateien im Arbeitsprofil zuzugreifen. Dabei wird immer die

Ausführungsgeschwindigkeit der jeweiligen Aktionen gemessen. Unterscheidet sich diese im Vergleich zu einer nicht vorhandenen Datei (oder Applikation) so liegt möglicherweise eine undichte Stelle vor, die erlaubt Dateien (oder Applikationen) zu erkennen.

Um nach aktiven Angriffsmöglichkeiten zu suchen wird eine erweiterte Version des aus [9] bekannten Frameworks verwendet. Ziel ist es hierbei, im Arbeitsprofil ausgeführte Aktionen zu erkennen. Im konkreten Beispiel wird versucht, mit Hilfe von im normalen Profil aufgenommenen Daten aufgerufene Webseiten im Arbeitsprofil voneinander unterscheiden zu können. Dazu müssen zuerst Methoden gefunden werden, welche auf die ausgeführten Aktionen reagieren. Deshalb werden in einem ersten Schritt alle aufrufbaren Methoden der Android API mehrmals automatisiert aufgerufen, während gleichzeitig Aktionen im Arbeitsprofil ausgeführt werden. Die Rückgabewerte der Methoden werden zwischengespeichert und dann am Ende dieser Phase verglichen. Unterscheiden sich die erhaltenen Rückgabewerte einer Methode, so wird diese Methode für eine genauere Untersuchung vorgesehen. Alle anderen Methoden, deren Rückgabewert sich nicht verändert, werden verworfen. Dadurch bleiben bei der eigentlichen Aufnahme der Rohdaten nur eine Handvoll Methoden über. Durch die wesentlich geringere Anzahl an aufzurufenden Funktionen kann jede Methode pro ausgeführter Aktion auch wesentlich öfter aufgerufen werden. Dies ermöglicht es, mehr Daten pro Aktion aufzunehmen.

Um zu zeigen, dass die Rückgabewerte einer Funktion Rückschlüsse auf die ausgeführten Aktionen ermöglichen, werden 20 verschiedene Aktionen ausgeführt. Jede einzelne Aktion wird dabei achtmal ausgeführt. Die Aktionen werden automatisiert ausgeführt. Dazu wird die Android Debug Bridge verwendet. Das heißt die Aktionen werden von einem PC aus gestartet, an dem das Smartphone angeschlossen ist. Nach der Aufnahme der Rückgabewerte wird versucht anhand der gesammelten Daten auf die ausgeführten Aktionen zurückzuschließen. Dazu wird der Dynamic Time Warping-Algorithmus verwendet. Je genauer die Rückgabewerte einer Funktion auf die ausgeführten Aktionen zurückschließen lassen, desto höher ist die Klassifizierungsgenauigkeit (classification accuracy).

4.1. Technische Umsetzung

Die Sammlung der Rohdaten erfolgt durch eine Applikation am Smartphone selbst. Der Aufnahmevorgang wird durch einen PC gesteuert. Die Kommunikation und Abstimmung erfolgt über die Android Debug Bridge (adb). Die Analyse der gesammelten Daten erfolgt nach Übertragung der Aufnahmen am PC.

Der Ablauf gestaltet sich wie folgt: Die Steuerungssoftware am PC startet die Aufnahmeapplikation am Smartphone und wartet bis sich diese initialisiert hat. Die Applikation am Smartphone lädt währenddessen alle benötigten Informationen und erstellt Objekte, die benötigt werden um die Methoden der Android API aufzurufen. Hierbei kann es dazu kommen, dass Objekte nicht erstellt werden können, weil beispielsweise benötigte Parameter nicht automatisiert erstellt werden können oder weil eine Klasse keinen öffentlichen Konstruktor hat. Diese werden bei der Initialisierung aussortiert. Returniert eine Methode jedoch ein Objekt anstatt eines primitiven Wertes oder eines Strings, so wird versucht diese Objekte zu verwenden um weitere Methoden aufzurufen. Dadurch können Methoden aus Klassen aufgerufen werden, welche nur über eine Methode abrufbar sind. Als Beispiel sei hier die Methode *getSystemService(String name)* der Klasse *android.content.Context* angemerkt, welche Objekte verschiedener Systemdienste bereitstellt, die nicht direkt über einen Konstruktor erstellt werden können. Durch die Verwendung der returnierten Objekte zum Aufruf von Methoden können so Dienste wie der *StorageManager* oder der *StorageStatsService*

untersucht werden. Sobald versucht wurde jede Methode aufzurufen und nicht aufrufbare Methoden verworfen wurden wird der Steuerungssoftware am PC gemeldet, dass das Laden der Aufnahmeapplikation abgeschlossen wurde.

Nach dem Laden der Applikation wird untersucht, welche der aufrufbaren Methoden potentiell auf ausgeführte Aktionen anspringen. Dazu werden von der Aufnahmeapplikation alle aufrufbaren Methoden mehrmals aufgerufen, während die Steuerungssoftware Aktionen im Arbeitsprofil des Smartphones ausführt. Um beispielsweise Webseiten zu öffnen, führt die Steuerungssoftware den folgenden Befehl aus:

```
adb am start --user 11 -a "android.intent.action.VIEW" -d
"https://www.a-sit.at/"
```

Dieser Befehl weist das Smartphone an, die Webseite https://www.a-sit.at/ im Arbeitsprofil (diese hat am Testgerät die Nutzerkennung 11) mit der Standardanwendung zu öffnen. Dies wird in der folgenden Evaluierung so lange gemacht, bis die Smartphoneapplikation jede Methode fünfzig Mal aufgerufen hat. Dadurch soll sichergestellt werden, dass jede Methode die Chance hat auf die ausgeführte Aktion zu reagieren und so keine Methode übersehen wird. Abschließend überprüft die Aufnahmeapplikation, ob sich die fünfzig gespeicherten Rückgabewerte jeder der untersuchten Methoden zumindest einmal unterscheiden. Ist dies der Fall, so wird die Methode weiter untersucht.

Ansonsten wird sie verworfen. Nach diesem Schritt meldet die Aufnahmeapplikation den Abschluss der Initialisierung an die Steuerungssoftware.

Nach der Initialisierung der Aufnahmeapplikation folgt die Aufnahme der eigentlichen Rohdaten. Dazu führt die Steuerungssoftware jede Aktion achtmal mit dem auch in der Initialisierung verwendeten Befehl aus. Anstatt jedoch wie in der Initialisierung möglichst viele Aktionen nacheinander auszuführen um die Methoden zur „Reaktion“ zu bringen, wird nach dem Ausführen einer Aktion sieben Sekunden gewartet bis die nächste gestartet wird. Dadurch soll sichergestellt werden, dass jede Aktion auch zu Ende geführt wird (also eine Webseite beispielsweise vollständig geladen wurde) und so auch erkannt werden kann. Dies wird für jede der 20 Aktionen in zufälliger Reihenfolge ausgeführt. Nach dem Abschluss der Aufnahme werden die gesammelten Rohdaten von der Steuerungssoftware auf den PC importiert. Die Rohdaten werden daraufhin analysiert.

5. Ergebnisse

In den folgenden zwei Sektionen werden die Ergebnisse der durchgeführten aktiven und passiven Experimente diskutiert.

5.1. Aktive Attacken

In Tabelle 1 sind die Resultate der Auswertung ersichtlich. Besonders über Netzwerkstatistiken lässt sich hierbei gut auf die aufgerufenen Webseiten zurückschließen. Die Klasse *android.net.TrafficStats* liefert hierbei sowohl die kumulierten gesendete als auch empfangenen Dateimengen. Ebenso liefert die Klasse die Anzahl der gesendeten und empfangenen Netzwerkpakete, welche eine vergleichbare Genauigkeit in der Erkennung ermöglicht wie die Anzahl an Bytes. Auch wenn die Statistiken nur geräteweit gesammelt werden, so erlauben sie dennoch die korrekte Klassifizierung eines Großteils der Aufrufe.

Tabelle 1 Methoden, die Rückschlüsse auf die geöffneten Webseiten ermöglichen.

Methode	Genauigkeit
android.net.TrafficStats_getTotalTxBytes()	90,6
android.net.TrafficStats_getTotalRxPackets()	86,25
android.net.TrafficStats_getTotalTxPackets()	85
android.net.TrafficStats_getTotalRxBytes()	83,1
android.os.storage.StorageManager_getAllocatableBytes(java.util.UUID)	71,25
android.app.usage.StorageStatsManager_getFreeBytes(java.util.UUID)	68,75
java.io.File_getUsableSpace()	68,75
java.io.File_getFreeSpace()	68,1
android.os.Process_getElapsedCpuTime()	25

Zusätzlich zu den Netzwerkstatistiken lassen auch Speicherstatistiken auf die aufgerufenen Webseiten zurückschließen. Diese sind sowohl über Android-spezifische Methoden der API, als auch der Java-API zugänglich. Zu den Android-spezifischen Methoden gehören *getAllocatableBytes(java.util.UUID)* der *android.os.storage.StorageManager* Klasse und *getFreeBytes(java.util.UUID)* der *android.app.usage.StorageStatsManager* Klasse. Erstere returniert die Anzahl an für die aufrufende Applikation verfügbaren Bytes auf dem Speichermedium, während zweitere die generelle Anzahl an noch zur Verfügung stehenden Bytes auf dem Speichermedium zurückgibt. Bei den Methoden in der Java-API handelt es sich um *getFreeSpace()* sowie getUsableSpace() der *java.io.File* Klasse. Die erste Methode liefert alle verfügbaren Bytes auf dem darunterliegenden Speichermedium des aktuell geöffneten File-Objektes. Hierbei werden etwaige Restriktionen bei der Berechnung des freien Speichers ignoriert. Die zweite Methode, *getUsableSpace()*, nimmt Berechtigungen und andere Restriktionen in die Berechnung auf und returniert nur die dem Aufrufer zur Verfügung stehenden Bytes. Über all diese Methoden lassen sich Rückschlüsse auf die geöffneten Webseiten ziehen. Die Genauigkeit dieser Methoden ist jedoch nicht ganz so hoch wie bei den Netzwerkstatistiken. Dennoch kann ein Großteil der Aufrufe korrekt eingeordnet werden. Details sind in Tabelle 1 ersichtlich.

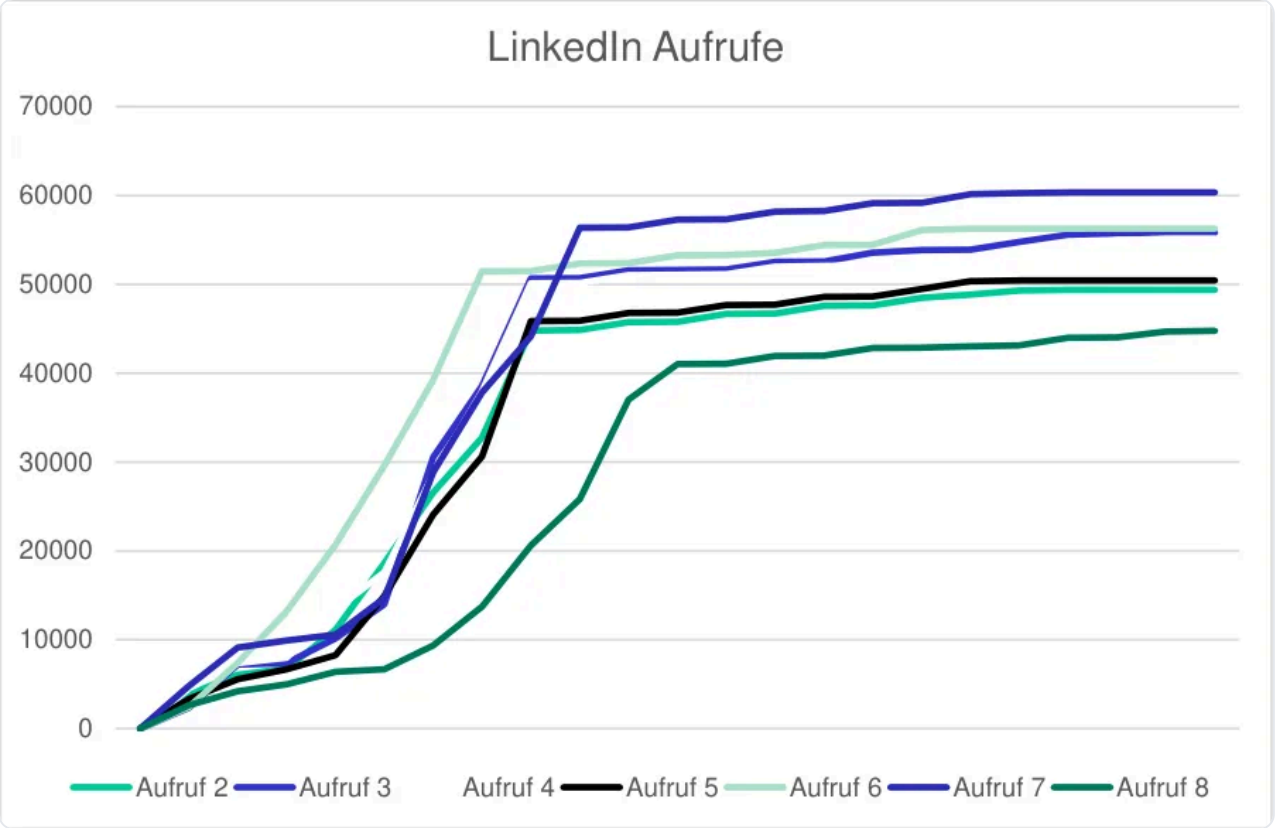


Abbildung 1: Gesammelte Daten der Methode getTotalTxBytes() der Klasse android.net.TrafficStats beim Aufruf von linkedin.com. Ein Auszug der gesammelten Rohdaten der Methode getTotalTxBytes() aus der Klasse android.net.TrafficStats sind in Abbildung 1 und Abbildung 2 ersichtlich. Diese Methode gibt die summierten gesendeten Bytes seit dem Gerätestart zurück. Zur besseren Übersichtlichkeit und Vergleichbarkeit wurden der erste gesammelte Wert von jedem dargestellten Wert subtrahiert. Dadurch beginnt jede gesammelte Aufnahme bei 0. Dies ermöglicht es die Aufnahmen zu vergleichen. Wie in den Abbildungen zu sehen ist, sind sich die gesammelten Rohdaten der jeweiligen Webseiten ähnlich. Sie unterscheiden sich jedoch leicht in der zeitlichen Achse, was primär durch unterschiedliche Netzwerklatenzen zu erklären ist. Vergleicht man die beiden Abbildungen so unterscheidet sich die Form der gesammelten Aufnahmen zwischen den beiden Webseiten substantiell. Auch die Anzahl an insgesamt gesendeten Bytes unterscheidet sich. Während bei vk.com pro Aufruf in etwa zwischen 40 und 45 Kibibyte gesendet werden, werden beim Aufruf von linkedin.com mit Ausnahme eines Ausreißers zwischen 50 und etwa 60 Kibibyte gesendet.

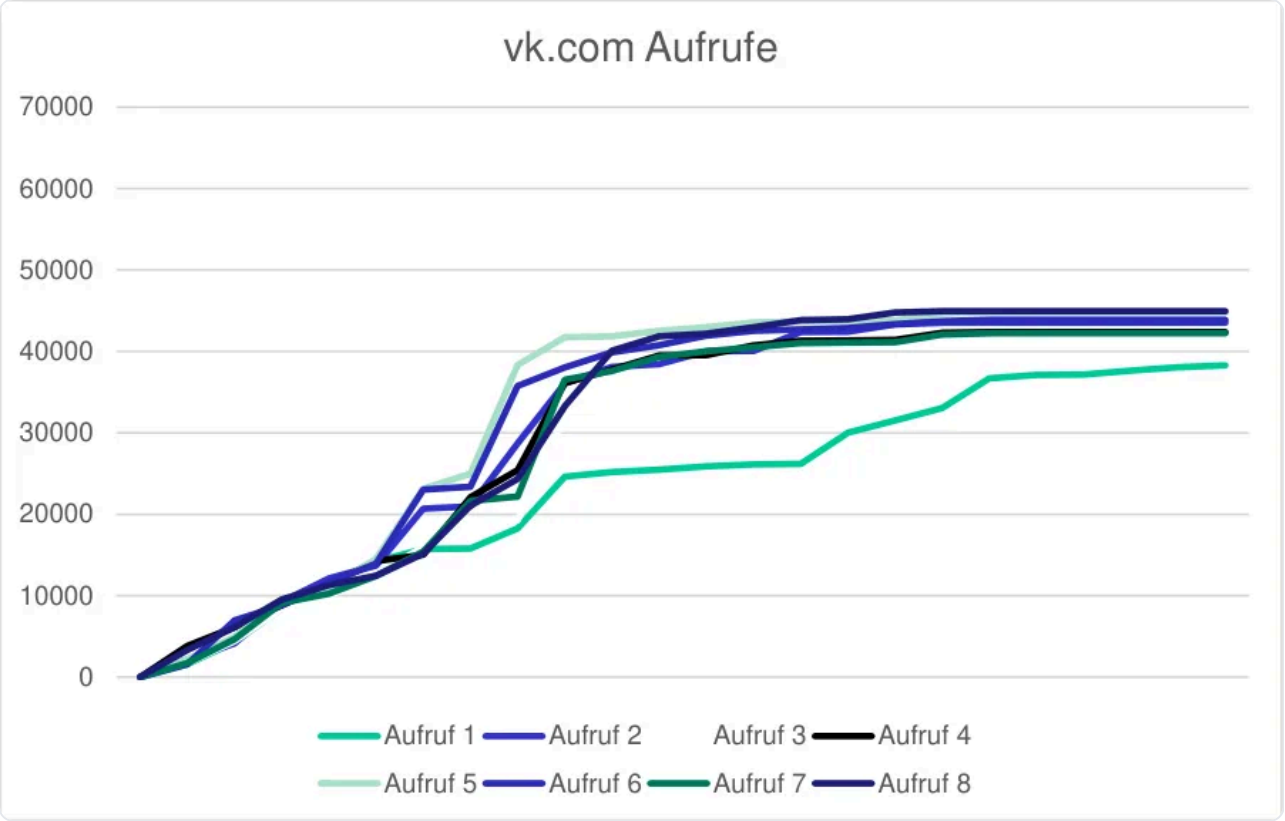


Abbildung 2: Gesammelte Daten der Methode getTotalTxBytes() der Klasse android.net.TrafficStats beim Aufruf von vk.com.

5.2. Passive Attacken

Bei den passiven Attacken wurde versucht, die nur im Arbeitsprofil installierte Applikation Outlook (*com.microsoft.office.outlook*) zu erkennen. Die Ausführungszeit der untersuchten Methoden unterschied sich jedoch nicht im Vergleich zum Aufruf mit Informationen einer nicht installierten Applikation. Auch die Existenz von Dateien im Arbeitsprofil konnte durch eine solche Attacke nicht festgestellt werden. Dies liegt daran, dass die Auflösung des Dateipfades an der Benutzergrenze aufhört. Alle im Arbeitsprofil gespeicherten Daten sind am Testgerät unter dem Pfad */storage/emulated/11* gespeichert (die des privaten Profils unter */storage/emulated/0*). Versucht man aus dem privaten Profil beispielsweise auf die am Testgerät existierende Datei */storage/emulated/11/Downloads/test.pdf* zuzugreifen, so bricht die Auflösung des Pfades bereits beim Ordner *11* ab. Dadurch dauert die Auflösung gleich lange, wie wenn man beispielsweise auf die nichtexistierende Datei */storage/emulated/11/Downloads/123.pdf* zuzugreifen versucht, da auch hier die Auflösung beim Ordner *11* abbricht, egal ob die Datei am Gerät existiert oder nicht.

6. Fazit

Durch eine aktive Attacke ist es für eine Applikation des privaten Bereichs möglich, Aktivitäten im Arbeitsprofil zu erkennen. Dazu muss die angreifende Applikation jedoch im Hintergrund laufen während die zu erkennende Aktion ausgeführt wird. Dies ist jedoch unter Android durchaus möglich, die Applikation muss nur eine unscheinbare Benachrichtigung im Infobereich anzeigen. In den Experimenten wurde gezeigt, dass durch die gesammelten Netzwerkstatistiken und Speicherstatistiken ein Großteil der geöffneten Webseiten erkannt werden konnte. Die Existenz von Dateien oder Applikationen konnte jedoch

nicht durch eine passive Attacke festgestellt werden. Hierbei verhindert die Isolation zwischen dem privaten Profil und dem Arbeitsprofil, dass auf die geschützten Informationen durch das System überhaupt zugegriffen wird. Dadurch ergeben sich auch keine Ausführungszeitunterschiede, welche von einem Angreifer erkannt werden könnten.

Referenzen

[1] Android Open Source Project, „Supporting Multiple Users | Android Open Source Project,“

[Online]. Available: https://source.android.com/devices/tech/admin/multi-user. [Zugriff am 02 09 2021].

[2] R. Mayrhofer, J. V. Stoep, C. Brubaker und N. Kralevich, „The Android Platform Security Model,“ *ACM Transactions on Privacy and Security,* p. 35, 2021.

[3] Google Developers, „Work Profiles | Android Developers,“ Google, [Online]. Available:

https://developer.android.com/work/managed-profiles. [Zugriff am 02 09 2021].

[4] Google Developers, „DevicePolicyManager | Android Developers,“ Google, [Online].

Available: https://developer.android.com/reference/android/app/admin/DevicePolicyManager.

[Zugriff am 02 09 2021].

[5] K. Siddiquie, N. Shafqat, A. Masood, H. Abbas und W. b. Shahid, „Profiling Vulnerabilities Threatening Dual Persona in Android Framework,“ *International Conference on Advances in the Emerging Computing Technologies,* 2019.

[6] Google Developers, „DevicePolicyManager | Android Developers | installSystemUpdate,“

2021. [Online]. Available: https://developer.android.com/reference/android/app/admin/DevicePolicyManager#installSystemUpdate(android.content.ComponentName,%20android.net.Uri,%20java.util.concurrent.Executor,%20android.app.admin.DevicePolicyManager.InstallSystemUpdateCallback). [Zugriff am 17 09 2021].

[7] T. Curran und R. d. Vries, „Exfiltrating Data from Managed Profiles in Android for Work,“

University of Amsterdam, Amsterdam, 2016.

[8] eLinux.org, „Android Binder,“ [Online]. Available: https://elinux.org/Android_Binder. [Zugriff am 28 09 2021].

[9] R. Spreitzer, G. Palfinger und S. Mangard, „SCAnDroid: Automated Side-Channel Analysis of Android APIs,“ *WiSec '18: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks,* pp. 224-235, 2018.