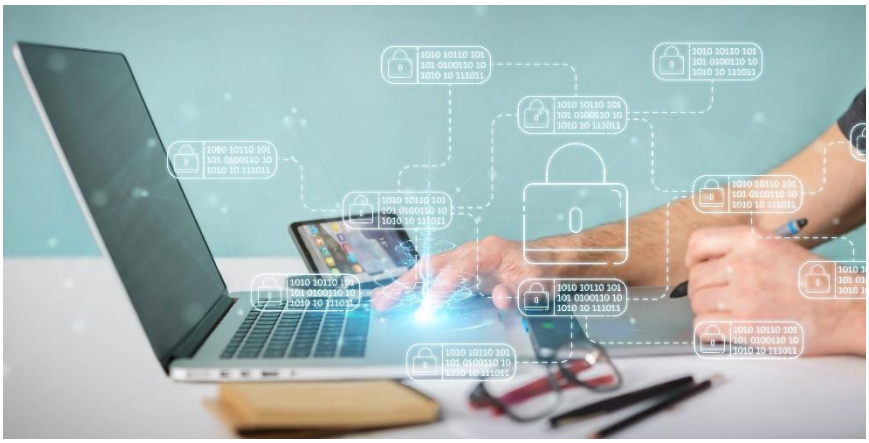
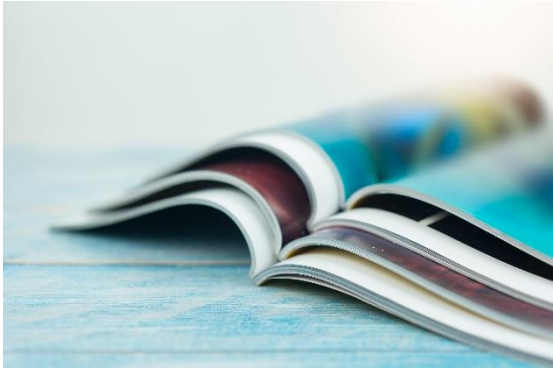
A-SIT

Zentrum für sichere Informationstechnologie - Austria

Reduzierung der Fingerprintbarkeit von Android-Smartphones





A-SIT

Reduzierung der Fingerprintbarkeit von Android-Smartphones

Autor:
Gerald Palfinger
Mail:
gerald.palfinger@iaik.tugraz.at

Datum: 30.11.2023

Abstract/Zusammenfassung:
In diesem Bericht wird eine Methode vorgestellt, welche es ermöglicht, die Fingerprintbarkeit von Android-Smartphones zu verringern. Dabei werden Informationsquellen in der Programmierschnittstelle so geändert, dass sich diese über verschiedene Geräte hinweg gleichen. Dazu wurde zuerst nach Informationsquellen gesucht, welche sich zwischen verschiedenen Geräten unterscheiden und es so potenziell erlauben, einen Fingerabdruck zu erstellen, um in weiterer Folge ein Gerät erneut und über verschiedene Applikationen hinweg zu identifizieren. Mit Hilfe der gefundenen Informationsquellen werden automatisch Patches generiert, welche die zurückgegebenen Werte der Informationsquellen an ein Referenzgerät anpasst, um dadurch die Fingerprintbarkeit zu reduzieren. Diese Patches können direkt auf Applikationen angewendet werden und benötigen so keine Änderung am Gerät selbst.

Inhalt

1. Einleitung

Smartphone-Applikationen sind zu unverzichtbaren Werkzeugen geworden, die eine breite Palette an Funktionen und Dienstleistungen bieten. Sie dienen dabei nicht nur der Unterhaltung, sondern ermöglichen auch die Kommunikation im Arbeitsalltag sowie den Zugriff auf eine Vielzahl von Applikationen, die das tägliche Leben erleichtern. Um diese Applikationen zu ermöglichen, gibt es eine umfangreiche Schnittstelle, die zum Programmieren von Applikationen verwendet werden kann. Neben dem Zugriff auf Funktionen des Gerätes bietet diese Schnittstelle auch den Zugriff auf verschiedene Informationsquellen. Während direkt identifizierbare Informationen entweder entfernt wurden oder eine Berechtigung durch den Nutzenden benötigen, kann durch die Kombination vieler Informationsquellen ein Fingerabdruck erstellt werden. In der Literatur wurde gezeigt, dass es sowohl auf Android als auch unter iOS eine Vielzahl an Informationsquellen gibt, welche solche Informationen bereitstellen [1] [2].

Um die Privatsphäre der Nutzerinnen und Nutzer zu verbessern, sollten die unterscheidbaren Merkmale zwischen den Geräten verringert werden, da dies es erschwert einen eindeutigen Fingerabdruck zu erstellen. Das ist Ziel dieses Projekts. Um die Programmierschnittstelle direkt am System zu ändern wäre jedoch ein weitreichender Eingriff in das System des Smartphones nötig, welcher aus Sicherheitsgründen nicht ohne Entsperrung des Bootloaders und zusätzlichem Rooten des Geräts möglich ist. Deswegen wurde der Ansatz gewählt potenziell fingerprintende Applikationen direkt zu patchen. Dies hat darüber hinaus den Vorteil, dass mögliche Auswirkungen auf die gepatchten Applikationen beschränkt bleiben. Zur Erstellung der Patches wurde folgender Ansatz gewählt. Zuerst wurden auf verschiedenen Geräten nach Informationsquellen gesucht, welche sich zwischen den untersuchten Geräten unterscheiden. Diese bieten potenziell Informationen, welche sich zur Erstellung eines Fingerabdrucks eignen können. Diese Quellen werden gesammelt, um dann im nächsten Schritt Patches zu entwickeln, welche die Information über die verschiedenen Geräte hinweg anpasst. Es wurde dazu eine Applikation entwickelt, welches die Patches automatisch aus den gesammelten Informationsquellen erstellt. Die Anwendung der Patches auf die Applikationen erfolgt mit Hilfe der Patch-Routine aus [3].

2. Hintergrund

In [2] wurde gezeigt, dass ähnlich wie im Browser auch unter Android durch die Kombination von Informationen ein eindeutiger Fingerabdruck eines Geräts erstellt werden kann. Dazu wurde von den Autoren manuell nach Informationsquellen gesucht, welche es ermöglichen könnten, ein Gerät zu fingerprinten. Als Ausgangspunkt wurden dabei die Dokumentation der Programmierschnittstelle gewählt. Ebenso wurde ausgehend von den am Gerät möglichen Einstellungen nach Wegen gesucht, um diese Informationen aus einer Applikation abzurufen. Abschließend wurde gezeigt, welche Informationsquellen verwendet werden können, um einen Fingerabdruck zu erstellen. Darüber hinaus wurden unter Android auch Anwendungen auf ihre Fingerprintaktivitäten untersucht. So wurde in [4] gezeigt, dass vor allem Tracking- und Werbebibliotheken eine Vielzahl an Informationsquellen abrufen, welche für die Erstellung eines Fingerabdrucks verwendet werden können.

Mit Android 10 [5] wurde für normale Anwendungen der Zugriff auf viele, in vorherigen Versionen noch zugängliche, eindeutige Identifizierungsmerkmale entfernt. Darunter fielen beispielsweise die Geräte-ID, verschiedene MAC-Adressen und die IMEI des Geräts. Systemanwendungen, also jene, die am Smartphone vorinstalliert sind, haben jedoch über privilegierte Berechtigungen weiterhin Zugriff auf diese Identifizierungsmerkmale. So benötigt beispielsweise der WLAN-Dienst weiterhin Zugriff auf die MAC-Adresse der WLAN-Schnittstelle, während der Telefon-Dienst Zugriff auf verschiedene für den Mobilfunk benötigte Informationen wie die IMEI hat. In [6] wurde untersucht, ob diese Dienste ausgenutzt werden können, um diese privilegierten Informationen preiszugeben. Hierbei konnte festgestellt werden, dass selbst unter Android 10 eine Methode weiterhin das eindeutige Identifizierungsmerkmal der ICCID zurückgibt. Da die Behebung per Sicherheitsupdate Probleme verursachte, wurde dieser Fehler erst mit Android 11 behoben. Während ansonsten keine weiteren Probleme in Android selbst erkannt wurden, wurden eine Vielzahl an Umgehungsmöglichkeiten auf Geräten anderer Hersteller gefunden. So konnten über gerätespezifische Anpassungen weiterhin auf eine Vielzahl an Identifizierungsmerkmale zugegriffen werden.

In [7] wurde systematisch untersucht, welche Informationsquellen verwendet werden können, um einen Fingerabdruck zu erstellen. Dazu wurde ein Framework erstellt, welches automatisch Informationsquellen in der Programmierschnittstelle abruft und zwischen verschiedenen Geräten vergleicht.

Informationsquellen, welche unterschiedliche Werte liefern, werden als fingerprintbar angenommen. Dabei wurde eine Vielzahl an Informationsquellen gefunden, die sich zur Erstellung eines Fingerabdrucks eignen. Diese inkludierten Methoden, Felder und auch Content Provider, welche alle über die Programmierschnittstelle zugänglich sind.

Auch unter iOS wurde gezeigt, dass die Erstellung eines Fingerabdrucks möglich ist [1]. Dabei wurden 29 Informationsquellen aus der Programmierschnittstelle durch die Autoren ausgewählt. Diese wurden dann mit Hilfe einer Applikation auf verschiedenen Geräten gesammelt, um einen Fingerabdruck dieser Geräte zu erstellen. In den Experimenten zeigten die Autoren, dass es möglich ist, durch erneutes Sammeln der Informationsquellen Geräte mit hoher Wahrscheinlichkeit anhand des vorher erstellten Fingerabdrucks wiederzuerkennen.

In [8] wurde eine Analyseumgebung erstellt, welche vormals unbekannte Bibliotheken erkennen soll, die Daten für das Fingerprinting von iOS-Geräten sammeln. Dazu wurden mehrere bekannte Fingerprinting-Bibliotheken analysiert. Die von diesen Bibliotheken verwendeten Informationsquellen wurden gesammelt und im Paper aufgelistet. Wie sich gezeigt hat, werden von vielen Bibliotheken nur Informationen aus einer kleinen Anzahl an Informationsquellen benötigt, um einen Fingerabdruck zu erstellen. Dadurch unterscheidet sich das Verhalten dieser Bibliotheken im Vergleich zum Browser-Fingerprinting, bei dem oft eine große Zahl an Informationen abgerufen wird. In einem weiteren Experiment wurde versucht, Fingerprinting-Aktivitäten in echten Applikationen zu erkennen. Dazu wird eine Frida-Analyseumgebung auf einem jailgebrokenem iPhone verwendet. Es wurde dabei angenommen, dass die Werte für das Fingerprinting hintereinander in einem kurzen Zeitfenster ausgelesen werden. Werden in einem Zeitfenster also genügend potenziell fingerprintbare Werte ausgelesen, so wird davon ausgegangen, dass die Applikation einen Fingerabdruck erstellt. Mit Hilfe dieser Vorgangsweise konnten weitere Informationsquellen ausgemacht werden, welche für das Fingerprinting verwendet werden. Darunter fallen beispielsweise verschiedene Bedienungshilfen oder andere Einstellungen und Geräte- sowie Betriebssystemeigenschaften.

Um einen umfangreichen Überblick an fingerprintbaren Informationsquellen zu bekommen, wurde auch für iOS eine Analyseframework geschaffen, welches solche Informationsquellen automatisch erkennt [9]. Ähnlich wie unter Android werden auch hier Methoden aufgerufen und deren Rückgabewerte gesammelt sowie Properties von Klassenobjekten ausgelesen. Dabei wurde festgestellt, dass hunderte Methoden und Objekte potenziell zur Erstellung eines Fingerabdrucks verwendet werden können.

In [10] wurde eine Erweiterung für Browser vorgestellt, welche die Fingerprintbarkeit von Firefox und Chrome verringern soll. Dazu wird die Programmierschnittstelle des Browsers eingeschränkt. Insbesondere werden Methoden und Eigenschaften eingeschränkt oder abgeändert, welche es erlauben, Informationen über das verwendete Gerät, die Browserinstanz, den Nutzer bzw. die Nutzerin sowie Informationen über die Umgebung zu eruieren. Ebenso können Sensorausgaben des Geräts sowie der Standort der Nutzenden simuliert werden.

3. Umsetzung

Um die Fingerprintbarkeit der Informationsquellen der Programmierschnittstelle von Android zu verringern, muss zuerst eruiert werden, welche Informationsquellen potenziell fingerprintbar sind. Dazu wurden verschiedene Geräte mit Hilfe eines automatisierten Tools verglichen und die sich unterscheidenden Informationsquellen wurden in einer Liste vermerkt. Anhand dieser Liste wurden Patches generiert, welche die Rückgabewerte der Informationsquellen ändern. Diese Patches wurden mit Hilfe eines erstellten Tools automatisch generiert. Schlussendlich müssen die erstellten Patches auf einzelne Applikationen angewendet werden. Auf diese drei Schritte wird in den folgenden Absätzen näher eingegangen.

3.1. Sammlung der Informationsquellen

Zur Umsetzung der Methode wird zuerst eine Liste an fingerprintbaren Informationsquellen sowie der vereinheitlichte Rückgabewert benötigt. Zur Erkennung dieser wird das Framework aus [7] verwendet. Um eine möglichst aktuelle Liste zu erhalten, wird auf Geräten mit aktuellem Android 14 getestet. Dabei werden sowohl Methoden der Programmierschnittstelle aufgerufen sowie Daten von Content Providern abgerufen. Die Rückgabewerte der Methoden sowie die abgerufenen Daten werden dabei gespeichert. Zur Entfernung von temporären Werten, welche sich nach einem Applikations- oder Geräteneustart ändern, wird die Sammlung pro Gerät zweimal ausgeführt. Die gesammelten und bereinigten Daten der Geräte werden dann untereinander verglichen. Daten, welche zwischen den Geräten unterschiedlich sind, auf dem Gerät selbst jedoch stabil, werden als fingerprintbar markiert.

Die gefundenen Informationsquellen werden in einer Liste aufbereitet. Diese enthält neben der genauen Signatur der Methode beziehungsweise dem Namen des Content Providers auch die auf den Testgeräten erhaltenen Werte. Diese Liste dient als Basis für die automatische Erstellung der Patches. Ein Beispiel für Elemente dieser Liste wird in Listing 1 gezeigt. Die Liste enthält jeweils drei Beispiele von Elementen aus Content Providern (startend mit `//`) sowie Rückgabewerte von Methoden (startend mit `public`). Dabei handelt es sich um den ausgewählten Benachrichtigungston, den verwendeten Autofill-Service (welcher Rückschlüsse auf den verwendeten Passwortmanager erlaubt), aktivierte Eingabemethoden, sowie die Größe von Icons, das Land und der verwendete Mobilfunkdienst.

```
//settings/system/notification_sound$notification_sound:::[('p6p',
['[content://media/internal/audio/media/68?title=Iapetus&canonical=1, false]']), ('p4',
['[content://media/internal/audio/media/171?title=Pixie%20Dust&canonical=1, false]'])]
//settings/secure$autofill_service:::[('p6p',
['[com.kunzisoft.keepass.libre/com.kunzisoft.keepass.autofill.KeeAutofillService, true]']), ('p4',
['[com.google.android.gms/.autofill.service.AutofillService, false]'])]
//settings/secure$enabled_input_methods:::[('p6p',
['[com.android.inputmethod.latin/.LatinIME;-1337596075;774684257:com.kunzisoft.keepass.libre/com.kunzisoft.keepass.magikeyboard.MagikeyboardService:com.simplemobiletools.keyboard/.services.SimpleKeyboardIME:org.kde.kdeconnect_tp/org.kde.kdeconnect.Plugins.RemoteKeyboardPlugin.RemoteKeyboardService:org.fcitx.fcitx5.android/.input.FcitxInputMethodService, true]']), ('p4',
['[com.google.android.inputmethod.latin/com.android.inputmethod.latin.LatinIME:com.google.android.tts/com.google.android.apps.speech.tts.googletts.settings.asr.voiceime.VoiceInputMethodService, true]'])]
public int
android.app.ActivityManager.getLauncherLargeIconSize():::[('p6p', ['168']), ('p4', ['126'])]
public java.lang.String
android.icu.util.ULocale.getDisplayName():::[('p6p',
['Deutsch (Österreich)']), ('p4', ['English (United States)'])]
public java.lang.String
android.telephony.TelephonyManager.getNetworkOperatorName():::[('p6p', ['educom']), ('p4', ['T-Mobile'])]
```

Listing 1: Beispiele an Unterschieden zwischen den untersuchten Geräten inklusive enthaltener Beispielwerte, dargestellt in Listenform.

3.2. Patch-Erstellung

Zur Erstellung der Patches wurde ein Programm entwickelt, welches die Patches automatisch erstellt. Dazu wird die im vorherigen Abschnitt gezeigte Liste an Informationsquellen, welche die auf den Geräten erhaltenen Werte beinhaltet, als Ausgangsbasis verwendet. Erkennt das Programm eine Methode durch das `public`-Attribut, so werden daraus die benötigten Informationen zum Ändern dieser Methode extrahiert. Dazu zählt im Besonderen der Typ des Rückgabewertes, der Name der Klasse, der Methodenname, und die Datentypen der Parameter der Methode. Zusätzlich wird erkannt, ob es sich um eine statische oder eine Instanzmethode handelt. Im Falle der Methode `getDisplayName()` aus Listing 1 wurde `java.lang.String` als Typ des Rückgabewertes extrahiert. Als Klasse wurde `android.icu.util.ULocale` und als Methodenname `getDisplayName()` erkannt. Da es keine Parameter gibt werden in diesem Fall keine Informationen über etwaige Parameter extrahiert. Bei der

betrachteten Methode handelt es sich um eine Instanzmethode, da kein `static`-Attribute in der Methodensignatur vorkommt.

Die durch das Programm extrahierten Informationen werden daraufhin verwendet, um den eigentlichen Patch zu erstellen. Dieser automatisch erstellte Patch ist für die betrachtete Methode `getDisplayName()` in Listing 2 ersichtlich. Da es sich um eine Instanzmethode handelt, wird der Patch-Methode zusätzlich das Instanzobjekt als Parameter übergeben. Dieses hat im gezeigten Beispiel den Namen `thiz`. Zusätzlich zur geänderten Methode wird aus dieser auch die Originalmethode aufgerufen. Dies wird gemacht, um etwaige gewünschte Änderungen am Instanzobjekt beziehungsweise Auswirkungen auf den Programmfluss auszulösen.

```
@PatchClass("android.icu.util.ULocale")
public class AndroidIcuUtilULocalePatches {


  @PatchInstanceMethod
  public static java.lang.String
getDisplayName(android.icu.util.ULocale thiz) {

OriginalMethods.android_icu_util_ULocale.getDisplayName();
    return "US";

  }
.
.
.
}
```

Listing 2: Ein Beispiel eines automatisch erstellten Patches. Dieser verändert die Methode `getDisplayName()` der Klasse `android.icu.util.ULocale`.

Um die über Content Provider verfügbaren Informationen abzuändern, werden die Methoden zum Abruf der Content Provider angepasst. Dadurch kommt auch die Anpassung der Content Provider ohne tiefe Eingriffe in die Systemdienste aus, welche die jeweiligen Content Provider bereitstellen. Content Provider werden anhand ihrer Content-URI startend mit `content://` identifiziert. Der Eintrag `autofill_service` aus Listing 1 ist hierbei beispielsweise über den `settings/secure` genannten Content Provider zugänglich. Die dazugehörige URI wäre also `content://settings/secure`. Um die Werte dieses Content Providers zu verändern, werden die Abrufmöglichkeiten in der Programmierschnittstelle angepasst. Content Provider können insbesondere über einen sogenannten `ContentResolver` abgerufen werden. Dabei wird der Methode `query(Uri,…)` des `ContentResolvers` die Content-URI mitgegeben. Diese Methode wird durch einen Patch überschrieben. Die gepatchte Methode reagiert auf Content-URIs, welche als fingerprintbar erkannt worden sind. Dazu prüft sie die übergebene URI gegen eine Liste von als fingerprintbar erkannter Content Provider. Verlangt eine gepatchte Applikation Zugriff auf eine solche Content-URI, so wird der Applikation anstatt des Verweises auf den fingerprintbaren Content Provider ein Verweis auf einen gepatchten Content Provider zurückgegeben. Dabei kann der ursprüngliche Content Provider komplett gegen einen gepatchten ausgetauscht werden. Diese Strategie wird beispielsweise bei Content Providern angewendet, welche Zugriff auf Systemeinstellungen bieten, da diese in der Regel immer von den Nutzenden

beeinflusst werden können. Ebenso ist es jedoch möglich, dass nur der Zugriff auf einzelne als fingerprintbar erkannte Werte abgeändert wird.

3.3. Patch-Anwendung

Die erstellten Patches aus dem vorherigen Schritt werden dann zu einem Patch-Paket zusammengefasst. Dieses kann dann auf einzelne Applikationen angewendet werden. Durch die Durchführung der Änderungen am Programmpaket ist keine Änderung an der Android-Laufzeitumgebung und damit auch keine Änderung am Gerät selbst notwendig. Bei der Umsetzung der Gegenmaßnahmen werden keine Programmierschnittstellen geändert, deren Nutzung der Nutzer bzw. die Nutzerin explizit durch das Erlauben einer Berechtigung erteilen muss. In dem erstellten Patch-Paket sind insgesamt 512 Patches für 162 Klassen vorhanden, welche auf die zu patchende Applikation angewendet werden. Nach Anwendung der Patches kann die gepatchte Applikation auf einem Gerät installiert werden.

3.4. Diskussion

Um die Fingerprintbarkeit zu reduzieren, wird im Wesentlichen versucht, ein Referenzgerät möglichst detailgetreu und konsistent nachzuahmen, ohne dabei die Nutzbarkeit von Applikationen einzuschränken. Manche der Änderungen können jedoch Auswirkungen auf die verwendeten Applikationen haben. So kann die Vereinheitlichung der Sprachumgebung Auswirkungen auf die in der Applikation angezeigten Sprache und Währungssymbole haben. Um dies zu umgehen, können einzelne Patches für betroffene Applikationen deaktiviert werden.

Um Änderungen an Applikationen anzuwenden, müssen diese neu gepackt und damit auch neu signiert werden. Da der für die Applikation verwendete Originalschlüssel dem Applikationsentwickler vorbehalten ist, muss diese erneute Signatur mit einem anderen lokal erzeugten Schlüssel durchgeführt werden. Dadurch kann es zu Problemen mit manchen Applikationen kommen. Dies ist vor allem dann der Fall, wenn Androids Key Attestation verwendet wird, um die Integrität der Applikation zu verifizieren. Während lokale Signaturüberprüfungen durch das verwendete Patching-Framework entfernt werden können, ist dies bei einer korrekt implementierten Remote-Attestation jedoch technisch nicht möglich. In diesem Fall kann das Framework also nicht erfolgreich eingesetzt werden. Eine solche Überprüfung wird laut [3] jedoch nur bei einer sehr kleinen Anzahl an Applikationen eingesetzt.

4. Fazit

In diesem Bericht wurde ein Framework vorgestellt, welches die Fingerprintbarkeit der Programmierschnittstelle von Android-Smartphones verringern kann. Dazu wurden zuerst potenziell fingerprintbare Informationsquellen durch den Vergleich verschiedener Smartphones mit Android 14 gesammelt. Aus diesen Geräten wurde ein Referenzgerät gewählt, auf das andere Geräte angepasst werden sollten. Dazu wurden automatisiert Patches erstellt, welche die Rückgabewerte von Methoden der Programmierschnittstelle an ein gewähltes Referenzgerät anpasst. Diese Patches können auf einzelne Programmpakete angewendet werden, wodurch das restliche System durch die Änderungen nicht beeinflusst wird. Manche der Änderungen können jedoch Auswirkungen auf die gepatchte Applikation haben, so können beispielsweise die Sprache von den Systemeinstellungen abweichen oder gewisse Funktionen nicht verfügbar sein.

Referenzen

[1] A. Kurtz, H. Gascon, T. Becker, K. Rieck und F. Freiling, „Fingerprinting Mobile Devices Using Personalized Configurations," in *Privacy Enhancing Technologies*, 2016.

[2] W. Wu, J. Wu, Y. Wang, Z. Ling und M. Yang, „Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers," *IEEE Access,* pp. 8073-8083, 2016.

[3] F. Draschbacher, „A2P2 - An Android Application Patching Pipeline Based On Generic Changesets," in *International Conference on Availability, Reliability and Security*, Benevento, Italy, 2023.

[4] C. F. Torres und H. Jonker, „Investigating Fingerprinters and Fingerprinting-Alike Behaviour of Android Applications," in *European Symposium on Research in Computer Security*, 2018.

[5] „Android Developers," [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes.

[6] M. H. Meng, Q. Zhang, G. Xia, Y. Zheng, Y. Zhang, G. Bai, Z. Liu, S. G. Teo und J. S. Dong, „Post-GDPR Threat Hunting on Android Phones: Dissecting OS-level Safeguards of User-unresettable Identifiers," in *NDSS*, 2023.

[7] G. Palfinger und B. Prünster, „AndroPRINT: analysing the fingerprintability of the Android API," in *International Conference on Availability, Reliability and Security*, Virtual, 2020.

[8] K. Heid, V. Andrae und J. Heider, „Towards detecting device fingerprinting on iOS with API function hooking," in *European Interdisciplinary Cybersecurity Conference*, Stavanger, Norway, 2023.

[9] G. Palfinger, „OCScraper: Automated Analysis of the Fingerprintability of the iOS API," in *SECRYPT*, Rome, 2023.

[10] L. Polčák, M. Saloň, G. Maone, R. Hranický und M. McMahon, „JShelter: Give Me My Browser Back," in *20th International Conference on Security and Cryptography, SECRYPT 2023*, Rome, 2023.