Batterie à bord : quand les jauges de carburant dépassent les limites

Vincent Giraud et David Naccache
prénom.nom@ens.fr

DIENS, École Normale Supérieure, Université PSL, CNRS
Ingenico

Résumé. La gestion de l'alimentation est cruciale pour les périphériques embarqués. Cette tâche est complexe en raison des incertitudes liées aux batteries qui les alimentent. Elle est souvent confiée à des circuits intégrés dédiés, appelés *jauges de carburant*, qui sont précis et efficaces, mais peuvent également être utilisés pour des activités malveillantes qui mettent en danger la confidentialité de la plateforme. Dans cet article, nous montrons comment l'isolation inter-applications peut être contournée sur Android, en nous concentrant sur la récupération d'un code personnel d'identification entré sur un autre processus.

1 Introduction

La vaste majorité des périphériques autonomes embarquent des batteries à base de lithium. Ces batteries offrent une grande densité d'énergie et une faible auto-décharge, sans effet mémoire. Cependant, leur comportement est difficile à analyser et à prévoir. La tension à leurs bornes n'est pas proportionnelle au niveau d'énergie restant, leur décharge est influencée par la consommation versatile et dynamique de la plateforme, la température et leur âge, et leur capacité totale est également influencée par ces paramètres. Pour ces raisons, la gestion de l'énergie sur un système embarqué est particulièrement complexe. De plus, les attentes des utilisateurs finaux ont augmenté ces dernières années, il n'est plus considéré acceptable de ne connaître son niveau de charge qu'au quart près, on attend une estimation au pourcentage près.

Assumer cette responsabilité peut donc représenter un fardeau, qui requiert bien du temps et du savoir-faire. Une possibilité est d'implémenter les opérations et les modélisations nécessaires au niveau du système d'exploitation, ou en tout cas de viser une exécution sur le processeur central. Or ce parti pris implique une charge supplémentaire afin que les calculs et les estimations puissent s'exécuter sur un composant déjà fortement sollicité. De plus, il est compliqué d'obtenir certaines mesures de manière fidèle depuis ce milieu : par exemple, la captation de la température sera

alors plus influencée par le calculateur lui-même que par la batterie. Avec cette implémentation, il devient également plus difficile d'estimer la qualité et l'âge de la source d'énergie. Enfin, ces défis s'accentuent si on doit gérer plusieurs batteries distinctes et séparées.

Afin de faciliter cet aspect, beaucoup de concepteurs de périphériques embarqués incluent des jauges de carburant.[1] Il s'agit de circuits intégrés qui sont dédiés à l'analyse et au suivi des métriques autour de la source d'énergie. Cela inclut, sans s'y restreindre, la tension de la source, le courant prélevé ou injecté vers elle, et la température. Pourtant digne d'intérêt dans le domaine de la sécurité, un aspect ignoré de ces composants est leur précision remarquable [3]. Cet avantage leur permet de produire leur autre fonctionnalité phare, l'estimation de l'âge, de la charge ou de la santé de la batterie.

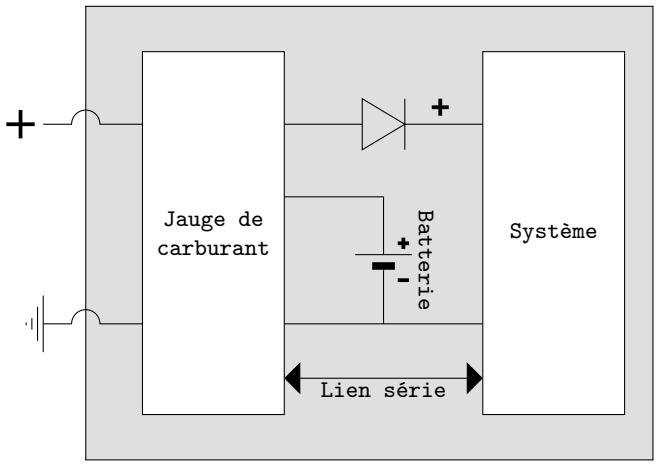


Fig. 1. Schéma représentant l'implémentation typique d'une jauge de carburant dans un système embarqué.

Ces circuits intégrés libèrent donc les concepteurs de périphériques et les développeurs de systèmes d'exploitation des responsabilités liées à la source d'énergie. Ils apportent des mesures plus réalistes car prélevées plus proche de la cible, et prennent en charge le calcul et l'algorithmique qui en découlent. Les logiciels destinés au système sur puce n'ont plus qu'à requêter les métriques ou données désirées ; ces demandes et leurs

[1] Désignées par *fuel gauge* en anglais. L'appellation *compteur coulomb* existe en français mais néglige la vaste majorité des fonctionnalités proposées par ces composants.

réponses transiteront via un lien série reliant la jauge de carburant et le système principal, comme illustré dans la figure 1. Ce canal correspond souvent à un bus de communication I^2C, géré par un pilote vivant dans l'espace noyau.

Cette délégation de responsabilité a notamment lieu dans des ordiphones, mais aussi dans des tablettes, et des consoles de jeu vidéo portables. Les jauges de carburant étant des circuits intégrés chers, on notera cependant qu'elles concernent essentiellement les produits visant le marché haut de gamme. Ceux destinés à une offre de prix plus contenue se contentent généralement de mesures et d'estimations peu précises.

Présence de jauge de carburant Avant achat, il est impossible de déterminer si un téléphone ou une tablette est équipé d'une jauge de carburant : la présence ou non de ce composant n'est pas indiquée sur les fiches techniques ou sur la documentation fournie par les constructeurs. Après achat, il est possible de vérifier sa présence en inspectant visuellement son circuit imprimé pour le trouver ou non. Sur Android, on peut logiciellement déterminer l'existence de cet outil dans le système via un terminal, en sondant les équipements liés à l'alimentation, sans nécessairement être root :

```
$ ls -a /sys/class/power_supply
```

Dans la liste résultante, on repère ainsi souvent des jauges de carburant dans la gamme Nexus et Pixel de Google, notamment dans le Pixel 6, où l'on reconnaît dans ce cas particulier le périphérique `maxfg` (`max` pour la marque Maxim Integrated, `fg` pour *Fuel Gauge*) :

```
battery
dc
gcpm
gcpm_pps
main-charger
maxfg
pca9468-mains
tcpm-source-psy-i2c-max77759tcpc
usb
wireless
```

1.1 Contexte logiciel : le cas d'Android

Le système d'exploitation Android repose sur un noyau Linux, avec un environnement utilisateur radicalement différent de celui habituellement

trouvé dans les distributions conventionnelles sur ordinateurs. Un choix décisif dans la conception d'Android a été d'assigner à chaque application un utilisateur Unix différent, permettant ainsi au système d'exploiter entre les applications l'isolation traditionnellement imposée entre les utilisateurs. Le module SELinux est déployé à partir de la version 4.3 pour renforcer cette politique. En dehors de l'espace noyau, au-dessus d'une couche minimale de bibliothèques et d'exécutables natifs, Zygote fait office de processus modèle pour l'instanciation d'applications : à chaque demande de la sorte, il se fork et change l'utilisateur associé au processus enfant pour respecter le paradigme évoqué. Les couches d'abstraction présentes sur un système Android sont illustrées de manière simplifiée dans la figure 2.

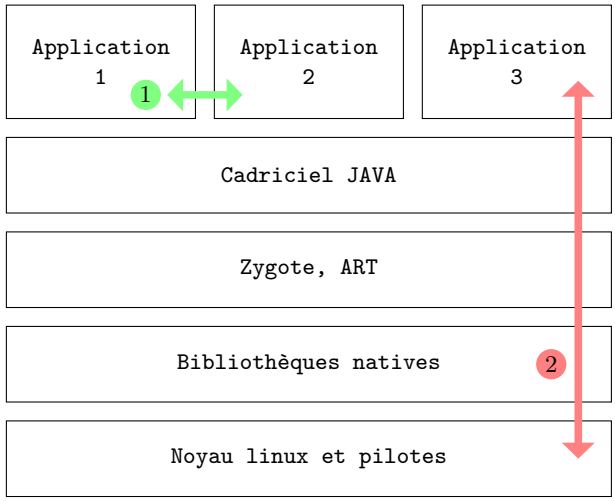


Fig. 2. Schéma d'abstraction simplifié d'un système Android

Ainsi, au niveau applicatif, les interactions entre applications sont limitées. En réalité, elles ne sont pas possibles directement ; et on peut éventuellement négocier des communications ou appels via Binder, le gestionnaire de communication inter-processus spécifique à Android. C'est ce composant qui prend en charge les appels entre services et activités, aspect cardinal dans ce système d'exploitation.

Si les accès horizontaux 1 sont régis par des règles claires, c'est moins le cas des interactions verticales 2, où une application sollicite des ressources matérielles présentes sur la plateforme. Dans le cas de téléphones et de

tablettes, celles-ci peuvent notamment inclure des périphériques comme un luxmètre, un accéléromètre, un gyroscope, un microphone, et une ou plusieurs caméras. La politique d'accès aux fonctionnalités associées est versatile, puisqu'elle est différente en fonction de la nature de la ressource, et du type d'interaction souhaitée. De plus, elle a souvent changé avec les versions d'Android, et elle peut être influencée par le constructeur de la plateforme. Exploiter ces ressources matérielles implique souvent de découvrir ce qu'on a le droit de faire avec un en particulier, sur un environnement précis.

L'accès au suivi énergétique est régi selon cette logique. L'éventualité d'un risque reposant dessus nous a alors incité à explorer cette voie.

1.2 Problématique

De par leur possible présence dans une brique essentielle des systèmes embarqués, il convient de procéder à une analyse de risques concernant les jauges de carburant. L'état de l'art de l'évaluation de la sécurité autour de ces circuits intégrés est inexistant. Tout porte à penser que les risques d'atteinte au matériel sont inexistants puisque ces composants n'ont pas de pouvoir de contrôle sur l'approvisionnement en électricité : ils sont à distinguer des *Power management integrated circuits*, ou PMICs. Les jauges de carburant pouvant exposer des mesures particulièrement précises, il convient plutôt d'étudier les risques vis-à-vis de la confidentialité, notamment sur des plateformes comme les téléphones et les tablettes, pouvant contenir une quantité substantielle d'informations personnelles.

2 Analyse des risques théoriques

2.1 Intégration des jauges de carburant dans les systèmes Android

Le service système `BatteryManager` existe depuis les débuts d'Android. Au commencement, il permettait seulement de connaître le statut de la batterie vis-à-vis de sa santé (`GOOD`, `OVERHEAT`, `DEAD`, `OVER_VOLTAGE`...) ou de son utilisation (`CHARGING`, `DISCHARGING`, `FULL`...), ainsi que, si applicable, la source de chargement (`USB` ou `AC`). Il s'est étoffé au fur et à mesure du temps, jusqu'à accueillir, dans la version 5.0 (dite *Lollipop*), des constantes afin de former des requêtes pouvant être redirigées vers une jauge de carburant.[2] On trouve notamment `CURRENT_NOW` pour obtenir

[2] `https://android.googlesource.com/platform/frameworks/base/+/refs/heads/lollipop-release/core/java/android/os/BatteryManager.java`

le courant instantané entrant ou sortant de la batterie en microampères, `CAPACITY` pour la capacité restante en pourcentage, et `ENERGY_COUNTER` pour l'énergie restante en nanowatt-heures.

Techniquement, une application, durant son exécution, peut invoquer le service `BatteryManager` et lui requêter n'importe lequel de ces attributs. Les informations en question seront récupérées en sondant les couches d'abstractions inférieures, et peut-être en consultant la jauge de carburant. Dans un premier temps, nous avons cherché à savoir si un contrôle était appliqué dans une des couches du système. La configuration de SELinux n'applique pas de restriction à ce sujet, bien qu'elle pourrait légitimement le faire. D'autres couches particulièrement portées à ce type de modération seraient celles du cadriciel Android en Java, ou de la machine virtuelle ART ; or il n'y a pas ici non plus de telle mesure. Comme on peut s'y attendre, les exécutables et bibliothèques natives présents sur le système n'y procèdent pas également. Après vérification sur la version 12 et antérieures, nous pouvons confirmer qu'Android ne bloque pas ces requêtes, quelles qu'elles soient, et quelle que soit l'application cliente.

Ce point peut d'ores et déjà représenter un problème pour l'utilisateur final, puisqu'il ne peut pas s'opposer au partage de ces informations. Lorsqu'une application met en place les moyens techniques pour les récupérer, fût-ce à des fins légitimes telles que l'économie d'énergie, on peut alors s'interroger sur l'utilisation réelle qui en est faite. L'entreprise Uber, cible de telles suspicions en 2016, a dû publiquement démentir ce genre d'exploitation.[3]

Accessoirement, notons également que certains navigateurs web permettent aux codes Javascript livrés par les sites de consulter le statut de la batterie et sa charge, via une interface du même nom, `BatteryManager`. Le moteur Javascript transmet ensuite la demande en suivant la même procédure qu'une autre application.

Un autre aspect nécessitant une attention particulière est la possibilité de capturer ces mesures en tout temps, y compris lorsque d'autres applications sont utilisées, ou lorsque le téléphone est en veille. Depuis Android 9 (dit *Pie*), il existe une permission `FOREGROUND_SERVICE`,[4] exigée par le gestionnaire d'activités lorsqu'une application demande à exécuter une tâche normalement en arrière plan. Comme la précédente, celle-ci est accor-

[3] `https://www.numerama.com/tech/170949-uber-sait-que-que-vous-paierez-plus-cher-si-votre-batterie-est-faible.html`

[4] `https://android.googlesource.com/platform/frameworks/base/+/refs/heads/pie-release/services/core/java/com/android/server/am/ActiveServices.java`

dée sans sollicitation de l'utilisateur. Cependant, son obtention implique d'avoir une notification présente au sein de la liste dédiée à cet effet, dans l'interface graphique du système. On trouve aujourd'hui de nombreuses applications qui nécessitent une notification permanente, afin de ne pas se voir sacrifiées par l'économiseur de batterie, ou pour pouvoir recevoir directement des communications sans passer par les services de Google. On peut ainsi envisager une usurpation, où une application présumément destinée au clavardage ou au jeu vidéo sonderait surtout, en réalité, la jauge de carburant en arrière-plan.

La considération venant ensuite consiste à évaluer la fréquence avec laquelle la jauge de carburant peut-être consultée. À partir d'Android 12, on constate dans le cadriciel Java l'apparition de la permission `HIGH_SAMPLING_RATE_SENSORS`.[5] Celle-ci a pour intention de limiter les scrutations au-delà de 200 Hz. Or, puisqu'il s'agit d'une permission normale, elle peut être réclamée sans avertissement visuel auprès de l'utilisateur. De plus, elle ne concerne de toute façon pas les jauges de carburant, comme le montre l'extrait présent en listing 1. Dans les versions antérieures à 12, cette mesure n'existe pas.

Listing 1: Extrait du gestionnaire de capteur système d'Android à partir de la version 12.

```
/**
 * Checks if a sensor should be capped according to
 ↪  HIGH_SAMPLING_RATE_SENSORS
 * permission.
 *
 * This needs to be kept in sync with the list defined on the native side
 * in frameworks/native/services/sensorservice/SensorService.cpp
 */
private boolean isSensorInCappedSet(int sensorType) {
    return (sensorType == Sensor.TYPE_ACCELEROMETER
            || sensorType == Sensor.TYPE_ACCELEROMETER_UNCALIBRATED
            || sensorType == Sensor.TYPE_GYROSCOPE
            || sensorType == Sensor.TYPE_GYROSCOPE_UNCALIBRATED
            || sensorType == Sensor.TYPE_MAGNETIC_FIELD
            || sensorType == Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED);
    }
```

En pratique, on constate que si Android transmet effectivement aussi rapidement qu'il le peut les requêtes de mesure, beaucoup de relevés consécutifs retournent la même valeur. On trouve l'explication dans la conception même des jauges de carburant : pour chaque métrique, elles

[5] `https://android.googlesource.com/platform/frameworks/base/+/refs/heads/android12-release/core/java/android/hardware/SystemSensorManager.java`

contiennent un registre physique qui est mis à jour avec une certaine fréquence. Si rien (à part les limites du lien série) n'empêche de scruter aussi rapidement qu'on le souhaite, les données relevées seront limitées par cette fréquence variant avec le modèle de circuit intégré. Sur le marché, on peut trouver des rafraîchissements autour de 4 et 10 Hz, ce qui disqualifie entre autres les attaques visant l'exécution de code cryptographique : l'attaque présentée dans [8] nécessite par exemple des mesures à l'échelle de la microseconde. Ces fréquences laissent néanmoins à portée les exploitations malveillantes sur les utilisations à vitesse humaine. On notera par ailleurs que même si la permission `HIGH_SAMPLING_RATE_SENSORS` mentionnée précédemment s'appliquait aux jauges de carburant, elle demeurerait inutile de par cette limitation inhérente au matériel.

2.2 État de l'art des exploitations de fuites d'information

La catégorie d'attaque la plus judicieuse dans ce cadre est celle des attaques par canaux auxiliaires. Elles reposent sur l'exploitation d'information provenant du fonctionnement du système, plutôt que sur des failles de conception, de spécification ou de protocole. Un exemple fondateur est la récupération de secrets pour Diffie-Hellman, RSA ou DSS en se basant sur les temps d'exécution [8]. En ce qui concerne les codes d'identification personnels, une attaque se basant sur les émissions électromagnétiques durant la vérification d'une séquence est présentée dans [9]. Sur les plateformes visées, les jauges de carburant offrent le potentiel d'exploiter un canal auxiliaire majeur, la consommation en temps réel, sans nécessiter d'équipement additionnel.

La saisie de code PIN sur téléphone a déjà été visée de plusieurs manières. Dans [4] et [11], elle est espionnée sur Android 2, via les capteurs de mouvement ou de rotation, et nécessite des données d'entraînement. Cette technique sera poussée dans [2], où les auteurs fusionnent les relevés sur plusieurs capteurs différents et de natures variées, en nécessitant encore un entraînement, présumément sur Android 5 ou 6. L'article [6] explique une technique d'espionnage applicable lorsque le téléphone est en train de charger via le lien USB : du matériel de captation spécifique, branché sur la ligne, intercepte le courant et infère la position des touchers à l'aide d'un réseau de neurones convolutifs, lui aussi entraîné mais uniquement par l'attaquant.

En exploitant les jauges de carburant, nous voulons proposer une attaque de code numérique ne nécessitant ni entraînement, ni scrutation d'une grande variété de capteurs embarqués.

2.3 Risques identifiés

En conséquence de cette intégration délétère des jauges de carburant, on identifie trois risques :

— Un premier mettant à mal la vie privée. Un attaquant peut journaliser à la seconde près des événements tels que l'utilisation ou non du téléphone, l'activation ou désactivation de connectivité sans fil, la réception ou émission de communication, etc.

— Un second créant un canal de communication caché sur la plateforme : les relevés de consommation en temps réel. On a vu qu'il serait accessible en lecture par toutes les applications. Il faut également considérer que tous les acteurs ont *de facto* un droit inaliénable à l'écriture dessus, puisque chacun peut, de par son exécution, provoquer une consommation moindre ou supplémentaire. Ainsi, par exemple, une application `A`, ayant accès à des données sensibles mais pas au réseau, pourrait les transmettre, au moyen de certaines techniques de modulation de signal, à une application `B`, pouvant accéder au réseau mais pas aux contenus sensibles.

— Un troisième, visant particulièrement les implémentations de solutions sécurisées : sur beaucoup de jauges de carburant, la fréquence de rafraîchissement, bien que basse, est du même ordre de grandeur que les interactions humaines. On peut alors redouter la récolte d'information durant l'entrée d'une donnée secrète.

Dans la suite de cet article, nous viserons en particulier le dernier, afin de récupérer un code d'identification personnel (PIN), destiné à un autre processus.

3 Exemple de récupération d'informations sensibles via la jauge de carburant

3.1 Moyens d'essai

Pour prototyper cette attaque, nous avons donc exploité les versions d'Android entre 9 et 12. Tout porte à penser que viser des versions antérieures ne posera aucun obstacle, hormis l'absence du support standard des jauges de carburant avant *Lollipop*. Dans cette section, nous nous focaliserons sur les appareils des gammes Nexus et Pixel de Google. Les essais se sont déroulés sans et avec un câble USB branché sur le plateforme. Dans le premier cas, la jauge de carburant retourne des valeurs négatives pour la consommation de courant instantanée (l'énergie sort de la batterie),

dans le second, le courant de charge est assez stable pour ne pas remettre en cause l'attaque, et on obtient des valeurs positives si le système consomme moins qu'il n'absorbe via le câble (il y a plus d'énergie qui rentre dans la batterie que ce qui en sort).

Tout d'abord, nous avons développé une simple application cible. Celle-ci, visible en figure 3, contient un champ attendant un code PIN, provoquant ainsi, lorsque touché, l'apparition d'un pavé numérique virtuel. Le besoin de confirmer son entrée avec une touche de validation dans l'angle de l'écran facilite une attaque temporelle, où l'on se base sur le laps de temps entre chaque appui. Par ailleurs, on constate que souvent, le pavé n'est pas exactement carré : la distance entre les touches 1 et 3 n'est pas la même que celle entre les touches 1 et 7. Ce facteur peut légèrement jouer en faveur de l'attaquant également. On notera surtout le fait que, dans la configuration de sortie d'usine de quasiment tous les téléphones, l'appui sur une touche provoque une vibration. Cette action ne peut être effectuée que par un composant mécanique nécessitant une énergie substantielle pour être activé, à savoir un moteur ou un électro-aimant. Elle représente donc forcément un agent aggravant dans cette situation.

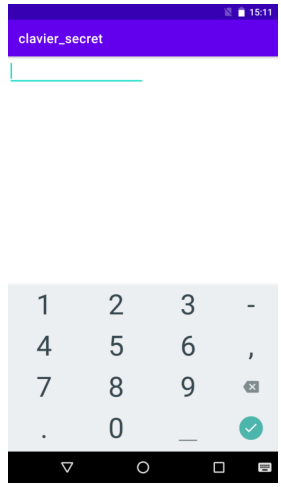
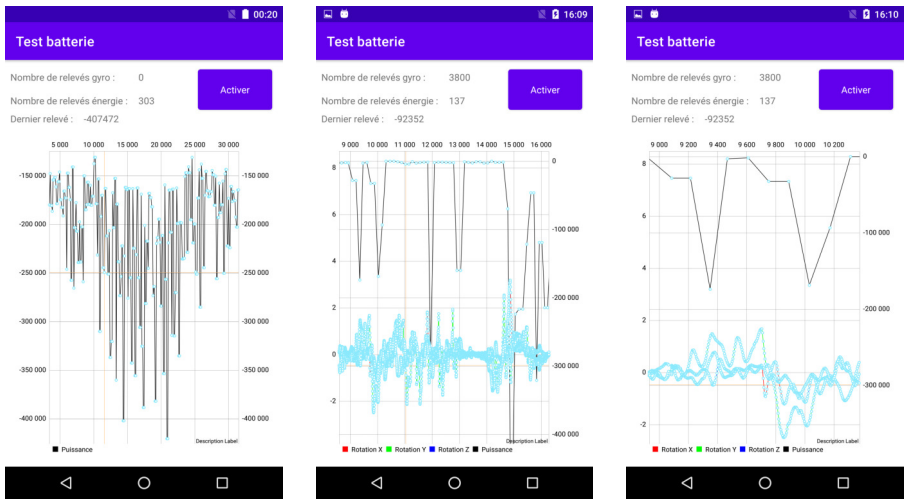


Fig. 3. Capture d'écran de l'application cible.

Dans un second temps, nous avons développé une application dédiée à l'attaque. En son sein, nous déclarons un service Android pour pouvoir scruter à notre souhait la jauge de carburant sans encombrer le processus

dédié à l'interface, et surtout afin de pouvoir le faire malgré le changement d'activité au premier plan, ou malgré la mise en veille du téléphone. Au fur et à mesure que l'on accumule des relevés énergétiques, on les gardera dans la mémoire tant que l'on sera en train de les consulter : essayer de les extraire en temps réel via un lien Android Debug Bridge (ADB) filaire ou pire, non filaire, risquerait d'amoindrir le rapport signal sur bruit. À des fins didactiques, on affichera ici directement un graphe exposant les mesures récoltées. Enfin, l'unique métrique sur laquelle nous nous baserons est la consommation de courant instantanée : la tension, la température ou la charge restante, bien que précises, ne seront pas utiles.

3.2 Résultats



(a) (b) (c)

Fig. 4. Captures d'écran de l'application en charge de l'attaque.

La figure 4(a) illustre une séquence où l'on a le doigt posé sur la dalle tactile entre les mesures 12 500 et 22 500, dénombrées sur l'axe des abscisses. La variance accrue vers le bas dans cette région indique que les jauges de carburant sont bien en mesure de détecter le delta de consommation électrique d'un téléphone ou d'une tablette lorsqu'on touche l'écran, y compris sans vibration. Utiliser un périphérique dont la dalle tactile est fissurée mais fonctionnelle ne change pas ce résultat. La courbe

baisse lors des touchers car la consommation augmente durant ces instants (il y a donc plus de courant qui sort de la batterie). Si on devine que ce différentiel de consommation est dû au phénomène physique en jeu sur les technologies capacitives, on peut également supputer qu'un traitement logiciel nécessaire pour gérer ce mode d'entrée soit également responsable.

Sur la capture de la figure 4(b), on peut voir une séquence typique correspondant à une saisie de code à 4 chiffres, avec validation. L'enclenchement de la vibration à chaque appui rend le relevé évident : on voit clairement à l'œil nu les pics correspondant à chaque entrée. Dans ce cas-là, il n'est donc pas nécessaire de déployer des techniques d'analyse de signal pour mener l'attaque. Cependant, la faible fréquence d'actualisation des jauges de carburant nous fait manquer de précision pour mener une attaque temporelle.

La capture de la figure 4(c) illustre le même jeu de donnée que sur celle du milieu, mais où l'on a agrandi les deux premiers pics. Les trois courbes concentrées au bas de l'écran correspondent aux relevés gyroscopiques, également prélevés. Ceux-ci peuvent nous permettre d'affiner les points de contact temporels : sous un pic énergétique, on peut retenir les points où les dérivés des trois axes de rotation atteignent zéro au même instant après une variation.

Une fois les délais entre les entrées précisément identifiés, on peut mener une attaque temporelle sur code. Nous avons développé un algorithme récursif et déterministe, qui, en fonction des laps de temps, déroule en partant de la fin l'arbre des codes possibles, comme illustré en listing 2. Fonctionner à l'envers est plus efficace puisque l'on sait qu'il est nécessaire pour l'utilisateur de confirmer son code en appuyant sur la touche de validation qui se situe à une position connue. On peut alors déterminer les numéros les plus susceptibles d'avoir été touchés juste avant, et ainsi de suite. De cette manière, dans l'arbre des codes possibles obtenu, la touche de validation représente la racine, et le premier chiffre des numéros considérés se situent aux extrêmités. Conceptuellement parlant, cette méthode est proche de celle présentée dans [5].

L'état de l'art montre que l'analyse temporelle peut fournir de bons résultats via l'exploitation de plusieurs techniques possibles, souvent appliquées aux bips sonores émis par les claviers numériques physiques. Dans [7], les codes PIN sont devinés en faisant usage de modèles de Markov cachés, alors que dans [10], l'utilisation de techniques d'apprentissage automatique est préférée. Dans tous les cas, quantifier les chances de succès ou la réduction du nombre de codes PIN possibles est compliqué, puisque ces valeurs dépendent des secrets eux-mêmes : une grande variabilité dans

la distance entre les touches facilite l'attaque, alors qu'une homogénéité la complique. Enfin, une réduction supplémentaire de l'espace des codes restant peut être obtenue en faisant une étude cinématique des relevés gyroscopiques, de toute façon récoltés pour affiner les pics. Par exemple, la légère rotation du périphérique nécessaire pour appuyer sur la touche 1 est différente de celle correspondant à l'appui sur la touche 0 : ce biais peut notamment aider à choisir le premier chiffre le plus probable.

Listing 2: Exemple de sortie de l'algorithme développé. Les codes proposés sont à l'envers, et la touche 10 correspond à la touche de validation utilisée en fin d'entrée.

```
(arbre '(2.52 2.01 2 1.74) (cons '(10) '()) 0)
=> ((((((10 6 8 2 3) (10 6 8 2 1))) (((10 6 2 8 9) (10 6 2 8 7))))))
```

3.3 Discussion

Dans ce travail nous avons démontré l'existence concrète d'un risque pour la confidentialité sur de nombreuses plateformes basées sur Android. Elle a été illustrée avec un exemple touchant à la récupération d'un code personnel, mais il convient de ne pas négliger ce danger en général, qui recouvre notamment l'espionnage d'activité et l'établissement ainsi que l'exploitation d'un canal de communication caché. Si un environnement logiciel est destiné à accueillir des contenus exécutables provenant de divers acteurs tiers, le concepteur du système doit accorder une attention particulière à l'intégration d'une jauge de carburant. La délégation de responsabilité vis-à-vis de la gestion de l'énergie peut en effet amener des considérations de sécurité additionnelles.

De même, si cet article a abordé essentiellement le cas d'Android de par sa vaste présence aujourd'hui, le risque présenté n'a rien de spécifique à ce système d'exploitation. Il existe sur le marché des périphériques exploitant une jauge de carburant à l'aide d'environnements différents. C'est notamment le cas de la Nintendo Switch, reposant sur un noyau FreeBSD, et bénéficiant d'un tel circuit intégré dans le cadre de sa gestion des batteries.

On peut aisément prendre en compte ce danger lorsqu'on maîtrise la plateforme et son système d'exploitation, puisqu'il suffit dans ce cas d'agir sur la politique de sécurité régissant les accès à de tels composants. Cette approche a été appliquée dans [1] pour réguler l'accès aux capteurs en général, au moyen de modifications sur les bibliothèques système natives, et sur les applications avant leur installation. La mitigation est cependant

bien plus complexe lorsque l'on est un acteur n'ayant accès qu'à la couche applicative, tel un développeur tiers. Une telle sécurisation de son processus sensible représente alors une tâche inédite, où l'on ne peut plus se reposer sur l'isolation inter-applications. De plus, la lutte contre les attaques par canaux auxiliaires est particulièrement complexe lorsqu'on travaille avec du code intermédiaire généré à partir de sources Java ou Kotlin, comme c'est majoritairement le cas sur Android. Ces travaux sont actuellement en cours d'étude.

4 Conclusion

Dans cet article, nous avons vu que certains périphériques autonomes embarquent une jauge de carburant et que celle-ci, de par ses capacités, peut impliquer des risques vis-à-vis de la confidentialité. Nous avons montré que leur inclusion dans le système Android y était vulnérable, en mettant en place une des exploitations possibles, à savoir l'espionnage de secret. Puisque ce danger vient contredire une garantie d'isolation normalement fournie par l'environnement, il entraîne des conséquences lourdes sur la production d'applications sensibles. Il devient alors nécessaire, pour les développeurs tiers, d'adopter des mesures et précautions. Ces solutions sont aujourd'hui à l'étude.

Remerciements

Les auteurs souhaitent remercier Guillaume Bouffard, de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), pour son accompagnement et son support tout au long de ce travail.

Références

1. Xiaolong Bai, Jie Yin, and Yu-Ping Wang. Sensor guardian : prevent privacy inference on android sensors. 2017.
2. David Berend, Bernhard Jungk, and Shivam Bhasin. There goes your PIN : Exploiting smartphone sensor fusion under single and cross user setting. 2017.
3. Mahmoud A. Bokhari, Yuanzhong Xia, Bo Zhou, Brad Alexander, and Markus Wagner. Validation of internal meters of mobile android devices, 2017.
4. Liang Cai and Hao Chen. TouchLogger : Inferring keystrokes on touch screen from smartphone motion. 2011.
5. Matteo Cardaioli, Mauro Conti, Kiran Balagani, and Paolo Gasti. Your PIN sounds good ! on the feasibility of PIN inference through audio leakage, 2019. Number : arXiv :1905.08742.

6. Patrick Cronin, Xing Gao, Chengmo Yang, and Haining Wang. Charger-surfing : Exploiting a power line side-channel for smartphone information leakage. 2021.
7. Denis Foo Kune and Yongdae Kim. Timing attacks on PIN input devices. In *Proceedings of the 17th ACM conference on Computer and communications security*. Association for Computing Machinery, 2010.
8. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology — CRYPTO '96*, 1996.
9. Hélène Le Bouder, Thierno Barry, Damien Couroussé, Jean-Louis Lanet, and Ronan Lashermes. A Template Attack Against VERIFY PIN Algorithms. In *SECRYPT 2016*, 2016.
10. Sourav Panda, Yuanzhen Liu, Gerhard Petrus Hancke, and Umair Mujtaba Qureshi. Behavioral acoustic emanations : Attack and verification of PIN entry using keypress sounds. 2020.
11. Zhi Xu, Kun Bai, and Sencun Zhu. TapLogger : inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, 2012.