FIGI

INITIATIVE MONDIALE EN FAVEUR
DE L'INCLUSION FINANCIÈRE

GROUPE DE TRAVAIL SUR LA SÉCURITÉ, L'INFRASTRUCTURE ET LA CONFIANCE

Audit de sécurité de différentes applications de services financiers numériques (DFS)

RAPPORT SUR L'AXE DE TRAVAIL SÉCURITÉ

Groupe de travail sur la sécurité, l'infrastructure et la confiance

Audit de sécurité de différentes applications de services financiers numériques (DFS)

12/2020

DÉNI DE RESPONSABILITÉ

L'Initiative mondiale en faveur de l'inclusion financière (FIGI) est un programme triennal mis en œuvre dans le cadre d'un partenariat entre le Groupe de la Banque mondiale, le Comité sur les paiements et les infrastructures de marché (CPMI) et l'Union internationale des télécommunications (UIT), et financé par la Bill and Melinda Gates Foundation. Il vise à faciliter et à accélérer l'application de réformes nationales en vue d'atteindre les objectifs nationaux en matière d'inclusion financière et, à terme, l'objectif mondial consistant à garantir un accès universel aux services financiers à l'horizon 2020. La FIGI finance des initiatives dans trois pays – la Chine, l'Égypte et le Mexique – et lutte contre trois obstacles distincts à l'accès financier universel, à travers le soutien qu'elle apporte aux trois groupes de travail suivants:

1) le Groupe de travail sur l'acceptation des paiements électroniques (dirigé par le Groupe de la Banque mondiale);
2) le Groupe de travail sur l'identité numérique pour les services financiers (dirigé par le Groupe de la Banque mondiale); et
3) le Groupe de travail sur la sécurité, l'infrastructure et la confiance (dirigé par l'UIT).

La FIGI organise trois colloques annuels rassemblant les autorités nationales, le secteur privé et d'autres parties prenantes compétentes afin de partager les nouvelles idées des groupes de travail et de faire le point sur l'avancée de la mise en œuvre au niveau national.
Le présent rapport a été élaboré par le Groupe de travail de la FIGI sur la sécurité, l'infrastructure et la confiance, dirigé par l'UIT. Les résultats, interprétations et conclusions exprimés dans ce rapport ne reflètent pas nécessairement les opinions des partenaires de la FIGI, notamment le CPMI, la Bill and Melinda Gates Foundation, l'UIT ou la Banque mondiale (y compris son Conseil d'administration ou les gouvernements qu'il représente). Les références éventuelles à certaines sociétés ou aux produits de certains fabricants ne signifient pas que l'UIT approuve ou recommande ces sociétés ou ces produits de préférence à d'autres de nature similaire, mais dont il n'est pas fait mention. Sauf erreur ou omission, les noms des produits propriétaires comprennent une lettre majuscule initiale. Les partenaires de la FIGI ne garantissent pas l'exactitude des données figurant dans le présent rapport. Les frontières, couleurs, dénominations et autres informations figurant sur les cartes de cet ouvrage n'impliquent aucune prise de position de la part des partenaires de la FIGI concernant le statut juridique d'un pays, d'un territoire, d'une ville ou d'une région ou de ses autorités, ni aucune reconnaissance ou acceptation de ces frontières.

À propos du présent rapport

Ce rapport a été rédigé par Sébastien Mathieu et Philippe Oechslin de la société Objectif Sécurité, avec l'aide de Vijay Mauree et Arnold Kibuuka.

Si vous souhaitez nous communiquer des informations complémentaires, veuillez contacter Vijay Mauree à l'adresse tsbfigisit@itu.int.

Table des matières

Résumé

Le Bureau de la normalisation des télécommunications (TSB) de l'UIT travaille à la mise en place d'un laboratoire d'audit de la sécurité des applications de services financiers numériques (DFS) sur les téléphones Android. Dans ce cadre, la société Objectif Sécurité a été mandatée pour réaliser un audit de la sécurité de plusieurs applications de DFS et pour définir un protocole d'audit des applications Android de DFS.

Une méthode d'essai fondée sur 18 tests a été mise au point, et trois applications de DFS ont été auditées suivant cette méthode. Ces 18 tests sont répartis en sept catégories tirées du célèbre classement des 10 principaux risques mobiles publié par l'Open Web Application Security Project (OWASP).

Un aperçu des résultats est fourni dans le tableau ci-dessous.

Les 10 principaux risques mobiles selon l'OWASP	Test		App1	App2	App3
M1 Utilisation détournée de la plate-forme	T1.1	Android:allowBackup	✓	✓	✓
	T1.2	Android:debuggable	✓	✓	✓
	T1.3	Android:installLocation	✓	✓	✓
	T1.4	Autorisations dangereuses	✓	✓	✓
M2 Stockage non sécurisé des données	T2.1	Android.permission.WRITE_EXTERNAL_STORAGE	×	✓	✓
	T2.2	Désactivation des captures d'écran	✓	✓	×
M3 Communications non sécurisées	T3.1	L'application doit uniquement fonctionner sur des connexions HTTPS	✓	✓	✓
	T3.2	L'application doit détecter les attaques par intercepteur utilisant des certificats non fiables	✓	✓	✓
	T3.3	L'application doit détecter les attaques par intercepteur utilisant des certificats fiables	✓	✓	×
	T3.4	Le manifeste de l'application ne doit pas autoriser le trafic en texte clair	✓	×	×
M4 Authentification non sécurisée	T4.1	Une authentification doit être requise pour accéder aux informations sensibles	✓	×	×
	T4.2	L'application doit se fermer automatiquement au bout d'un certain délai d'inactivité	✓	✓	✓
	T4.3	Si une nouvelle empreinte digitale est ajoutée, l'authentification existante par empreintes digitales doit être désactivée	✓	✓	✓
	T4.4	Les demandes sensibles ne peuvent pas être répétées	✓	×	✓
M5: Cryptographie insuffisante	T5.1	L'application ne doit pas utiliser de primitives cryptographiques non sécurisées	×	×	×
	T5.2	Les connexions HTTPS doivent être configurées conformément aux bonnes pratiques	✓	✓	✓
	T5.3	L'application doit chiffrer les données sensibles envoyées via des connexions HTTPS	×	✓	✓
M8: Falsification de code	T8.1	L'application ne doit pas pouvoir être exécutée sur un appareil rooté	×	✓	✓
M9: Rétro-ingénierie	T9.1	Le code de l'application nécessite une obfuscation	✓	✓	✓

Abréviations

CA	Autorité de certification
DES	Data Encryption Standard
DFS	Services financiers numériques
ECB	Dictionnaire de codes
HTTPS	Protocole de transfert hypertexte
MD	Synthèse de message
MITM	Attaque par intercepteur
OWASP	Open Web Application Security Project
PIN	Numéro d'identification personnel
PUK	Clé de déverrouillage personnelle
RC	Chiffrement Rivest
SHA	Algorithme de hachage sécurisé
SSL	Couche de connexion sécurisée
TLS	Sécurité de la couche de transport

Audit de sécurité de différentes applications de services financiers numériques (DFS)

1 À PROPOS DES APPLICATIONS

Les trois applications de DFS analysées ont été sélectionnées selon les critères suivants: deux d'entre elles sont développées par des fournisseurs présents sur le marché africain, et la troisième est une application de DFS européenne. Dans ce rapport, ces trois applications de DFS sont désignées respectivement comme App1, App2 et App3 et les résultats de leur audit de sécurité ont été anonymisés.

1.1 App1

Application de paiement mobile basée en Europe, App1 relie la carte de crédit et le compte bancaire de l'utilisateur. Elle peut être utilisée pour envoyer, demander ou recevoir des fonds. Cette application peut également être utilisée pour effectuer des achats en ligne (en scannant des codes QR) ou pour réaliser des paiements sans espèces dans diverses situations: en magasin, au restaurant, sur des parcmètres utilisant des codes QR ou via des beacons (balises connectées) installés en vitrine. Pour créer un compte, les utilisateurs ont besoin d'un numéro mobile et des coordonnées d'une carte de crédit ou d'un compte bancaire.

1.2 App2:

App2 est gérée par un opérateur de réseau mobile qui propose des services financiers numériques (DFS) dans différentes parties d'Afrique. Application de services financiers mobiles innovante, elle permet à ses utilisateurs d'envoyer des fonds localement et à l'international, d'effectuer des transactions de biens ou services partout dans le monde, et d'opérer des virements entre leur portefeuille mobile et leur compte en banque. Pour créer un compte, l'utilisateur doit avoir un numéro mobile avec l'opérateur. Les utilisateurs de l'application n'ont pas besoin d'avoir de compte bancaire.

1.3 App3:

App3 est également gérée par un opérateur mobile, présent cette fois dans plusieurs pays d'Afrique et d'Asie. Elle permet à ses utilisateurs d'envoyer des fonds à leurs contacts, de payer des biens ou services, ainsi que de réaliser des virements entre leur portefeuille mobile et leur compte en banque. Pour créer un compte, l'utilisateur doit avoir un numéro mobile avec l'opérateur. Les utilisateurs de l'application n'ont pas besoin d'avoir de compte bancaire.

2 MÉTHODE DE TEST

L'objectif des tests est de noter le niveau de sécurité de diverses applications de DFS pour smartphone, suivant une grille normalisée. Pour ce faire, les applications testées sont installées sur un téléphone témoin et leurs caractéristiques de sécurité sont analysées au moyen d'une série d'outils de test. Les tests ont été choisis de manière à pouvoir être réali-

sés relativement facilement, à l'aide d'outils à code source ouvert.
Ces tests reprennent la typologie des 10 principaux risques mobiles établie par l'OWASP. L'OWASP[1] est une fondation à but non lucratif qui travaille à améliorer la sécurité des applications Web. L'un de ses projets est le classement des 10 principaux risques mobiles[2], qui met en exergue les risques de sécurité suivants:

a) M1 Utilisation détournée de la plate-forme
b) M2 Stockage non sécurisé des données
c) M3 Communications non sécurisées
d) M4 Authentification non sécurisée
e) M5 Cryptographie insuffisante
f) M6 Autorisation non sécurisée
g) M7 Qualité du code client
h) M8 Falsification de code
i) M9 Rétro-ingénierie
j) M10 Fonctionnalité externe

Les catégories M6, M7 et M10 n'entrent pas dans le champ de nos tests, puisque leur audit impliquerait l'accès au code source des applications et la rétro-ingénierie de leur structure.
Les 18 tests suivants ont été retenus pour leur pertinence et leur faisabilité:

2.1 M1 Utilisation détournée de la plate-forme

Ces tests consistent à analyser le manifeste de l'application. Les points vérifiés sont les suivants:

a) T1.1 Android:allowBackup[3]:
   Ce paramètre doit être réglé sur "false", qui n'est pas la valeur par défaut.
   Si cet attribut est réglé sur "false", aucune sauvegarde ou restauration de l'application ne sera jamais réalisée, même au travers d'une sauvegarde de l'ensemble du système qui permettrait de sauvegarder toutes les données de l'application.
b) **T1.2 Android:debuggable[4]:**
   Ce paramètre doit être réglé sur "false", qui est la valeur par défaut.
   Si une application est définie comme débogable, des intrus peuvent injecter leur propre code pour l'exécuter dans le contexte de processus d'application vulnérable.
c) **T1.3 Android:installLocation[5]:**
   Ce paramètre doit être réglé sur "internalOnly", qui est la valeur par défaut, ou bien être désactivé.
   Si ce paramètre est réglé sur "auto" ou "preferExternal", l'application peut être installée sur une carte mémoire amovible.
   En accédant à la carte mémoire, un attaquant peut falsifier le code de l'application ou en extraire des informations sensibles.
   N.B.: Même si l'application échoue à ce test, cela ne signifie pas nécessairement qu'elle sera installée sur la carte amovible.
d) **T1.4 Autorisations dangereuses:**
   L'application ne doit pas solliciter d'autorisations dangereuses sans raison valable.
   Les applications Android doivent explicitement solliciter des autorisations pour de nombreux types d'opérations. Android qualifie certaines de ces autorisations de "dangereuses". Pour toute autorisation dangereuse, l'application doit explicitement solliciter l'aval de l'utilisateur en affichant un message dans une boîte de dialogue (ex.: Autoriser l'application à passer des appels?)
   Il existe certaines raisons valables de solliciter des autorisations dangereuses. Par exemple, une application de DFS qui a besoin de scanner des codes QR pour effectuer des paiements devra solliciter l'autorisation d'utiliser la caméra.
   N.B.: Les autorisations relatives au stockage de données sont traitées dans la section suivante.
   Une application qui requiert des autorisations dangereuses peut se servir de ces autorisations pour attaquer l'utilisateur. Si elle obtient l'autorisation de passer des appels, elle peut par exemple appeler des numéros surtaxés.

2.2 M2 Stockage non sécurisé des données

Ces tests consistent à analyser le manifeste de l'application et à tester l'application sur un téléphone portable. Les points vérifiés sont les suivants:

a) **T2.1 Android.permission.WRITE_EXTERNAL_STORAGE:**
   L'application ne doit pas solliciter cette autorisation sans raison valable.
   Cette autorisation permet à l'application de lire et d'écrire des données sur une carte mémoire insérée dans le téléphone. Si l'application a besoin de stocker d'importants volumes de données non sensibles, il est alors justifié d'écrire ces données sur un dispositif de stockage externe.
   En accédant à la carte mémoire, un attaquant peut falsifier le code de l'application ou en extraire des informations sensibles.

N.B.: Même si l'application échoue à ce test, cela ne signifie pas nécessairement qu'elle écrira des données sensibles sur un dispositif de stockage externe.

b) **T2.2 Désactivation des captures d'écran:**
L'application ne doit pas permettre les captures d'écran lorsqu'elle est en fonctionnement et doit uniquement afficher un fond blanc dans le sélecteur de tâches.
Il s'agit là d'un comportement typique des applications sécurisées, qui peut être réalisé à l'aide d'un paramètre applicatif appelé FLAG_SECURE[6]. Cet aspect peut être testé en lançant l'application pour 1) essayer de réaliser une capture d'écran et 2) vérifier la vignette de l'application dans le sélecteur de tâches.
Si ce paramètre n'est pas activé, une application malveillante peut dérober des informations sensibles s'affichant sur l'écran de l'application.

2.3 M3 Communications non sécurisées

a) **T3.1 L'application doit uniquement fonctionner sur des connexions HTTPS:**
En auditant le trafic de l'application et en observant les paquets, on ne devrait observer que du trafic HTTPS.
Le trafic HTTPS est chiffré. Même s'il existe d'autres manières de chiffrer le trafic d'une application, le recours à une connexion HTTPS est le moyen le plus couramment utilisé pour assurer la communication de données entre une application et un serveur. Si les données sont transmises via une connexion HTTP ou un autre protocole non chiffré, un attaquant peut alors facilement les intercepter ou même les modifier.

b) **T3.2 L'application doit détecter les attaques par intercepteur utilisant des certificats non fiables:**
Si les données transitent via un proxy "machine-in-the-middle" (MITM) qui ne dispose pas de certificat fiable pour le serveur de l'application, l'application doit refuser la connexion.
Les proxys MITM peuvent être utilisés pour intercepter des données circulant en HTTPS afin de les déchiffrer pour inspection et modification, puis de les chiffrer à nouveau et de les envoyer vers le serveur ciblé. Comme la plupart des attaquants ne disposent pas de certificat valide auprès du serveur de destination, l'application doit détecter les proxys dont le certificat n'est pas délivré par une autorité de confiance. Si l'application ne vérifie pas la validité du certificat, un attaquant peut intercepter et modifier le trafic.

c) **T3.3 L'application doit détecter les attaques par intercepteur utilisant des certificats fiables:**
Si les données transitent via un proxy MITM qui utilise un certificat délivré par une autorité de certification à laquelle le smartphone fait confiance, l'application doit refuser la connexion. Lorsque l'opérateur du proxy est capable de générer des certificats auxquels le téléphone fait confiance, différents cas de figure peuvent se présenter. L'opérateur peut être une autorité de certification (par exemple un gouvernement) ou une entreprise qui a installé son certificat racine sur les téléphones qu'elle produit. Le certificat racine peut également avoir été installé à la main par l'utilisateur ou par un attaquant. L'application peut se prémunir de ce type d'attaque en épinglant le certificat racine. L'application connaît alors l'autorité de certification censée délivrer le certificat du serveur et elle refusera les certificats délivrés par d'autres autorités, même s'il s'agit d'autorités de confiance. L'exécution de ce test suppose généralement de rooter le téléphone afin de pouvoir installer un certificat racine.
Si l'application n'épingle pas de certificat, le trafic peut alors être intercepté par des gouvernements ou par des attaquants qui ont réussi à pirater l'une des nombreuses autorités de certification racine existantes.

d) **T3.4 Le manifeste de l'application ne doit pas autoriser le trafic en texte clair:**
À partir d'Android 8.1, le trafic en texte clair est désactivé par défaut. Le manifeste de l'application ne doit pas contenir de paramètres qui supplantent ce mode par défaut. Il peut s'agir du paramètre "android:usesCleartextTraffic" de l'application ou du paramètre "clear textTrafficPermitted" dans la configuration de la sécurité du réseau.
Lorsque le trafic en texte clair est désactivé, l'application et les autres composantes qu'elle utilise (lecteur multimédia, par exemple) bloqueront tout trafic en texte clair.
Le trafic en texte clair peut facilement être espionné et manipulé par des auteurs d'attaques.
N.B.: Même si l'application échoue à ce test, cela ne signifie pas nécessairement qu'elle enverra ou recevra du trafic en texte clair.

2.4 M4 Authentification non sécurisée

Les tests suivants s'effectuent en lançant l'application sur un téléphone et en observant son comportement.

a) **T4.1 Une authentification doit être requise pour accéder aux informations sensibles:**
L'application doit demander un mot de passe, un code PIN ou une empreinte digitale avant de donner accès à des fonctionnalités ou à des informations sensibles (paiements et soldes de compte bancaire, par exemple).
On peut tester cela en lançant l'application sur un téléphone.
Si l'application ne demande pas à l'utilisateur de s'identifier à chaque nouvelle connexion, il est possible qu'un attaquant vole ou emprunte un téléphone déverrouillé et accède ainsi à des fonctionnalités ou à des informations sensibles.

b) **T4.2 L'application doit se fermer automatiquement au bout d'un certain délai d'inactivité:**
Pour tester cela, on laisse l'application ouverte pendant un certain temps et l'on observe si elle s'éteint automatiquement.
Si l'application ne s'éteint pas automatiquement au bout d'un laps de temps défini ou si ce laps de temps est trop long, il est possible qu'un attaquant vole ou emprunte un téléphone déverrouillé et accède ainsi à des fonctionnalités ou à des informations sensibles.

c) **T4.3 Si une nouvelle empreinte digitale est ajoutée, l'authentification existante par empreintes digitales doit être désactivée:**
Lorsqu'une nouvelle empreinte digitale est enregistrée sur le téléphone, l'application doit empêcher l'authentification par empreinte digitale jusqu'à ce que l'utilisateur ait entré le code PIN ou le mot de passe pour accéder à l'application.

Sinon, le risque est qu'un attaquant parvienne à enregistrer sa propre empreinte sur le téléphone et puisse ainsi accéder aux applications protégées par empreinte digitale.

d) **T4.4 Il ne doit pas être possible de répéter les demandes interceptées:**
La répétition d'une demande (de transfert de fonds, par exemple) captée par un proxy ne doit pas déboucher sur une nouvelle exécution de cette demande.
Le risque est qu'un attaquant qui intercepte une demande de transfert de fonds puisse répéter cette demande afin de voler de l'argent à la victime.

2.5 M5: Cryptographie insuffisante

a) **T5.1 L'application ne doit pas utiliser de primitives cryptographiques non sécurisées:**
Les algorithmes tels que MD5, SHA-1, RC4, DES, 3DES, Blowfish, le mode ECB pour les chiffrements par blocs, les générateurs aléatoires non cryptographiques sont connus pour être faibles et ne doivent pas être utilisés par l'application[7].
On peut tester cela en analysant le code binaire de l'application afin de déterminer si elle a recours à ces algorithmes non sécurisés.
Si des informations sensibles sont transmises via ces algorithmes, il existe alors un risque qu'un attaquant les espionne ou les manipule. Le fait que ces algorithmes soient utilisés ne signifie pas forcément qu'ils le sont dans le cadre d'opérations sensibles. Néanmoins, pour éviter tout soupçon, il est recommandé de ne pas utiliser ces algorithmes.
N.B.: Même si l'application échoue à ce test, cela ne signifie pas nécessairement qu'elle protège ses données sensibles au moyen de primitives cryptographiques non sécurisées.

b) **T5.2 Les connexions HTTPS doivent être configurées conformément aux bonnes pratiques:**
L'observation du trafic du réseau de l'application permet d'identifier les serveurs avec lesquels elle communique. La configuration HTTPS de ces serveurs peut être testée à l'aide d'un outil comme Qualys SSL Labs[8]. La note globale doit être égale ou supérieure à B.
Si la connexion HTTPS n'est pas correctement configurée, il existe un risque que les données soient espionnées ou manipulées à des fins malveillantes.

c) **T5.3 L'application doit chiffrer les données sensibles envoyées via des connexions HTTPS:**
Cet aspect peut être testé en interceptant le trafic à l'aide d'un proxy MITM (voir tests en M3).
N.B.: Si l'application épingle les certificats, il est nécessaire de désactiver cette protection afin de pouvoir intercepter le trafic. Cela n'est pas toujours possible.
Si l'application elle-même ne chiffre pas les données, un intercepteur peut alors les espionner ou les modifier.

2.6 M8 : Falsification de code

T8.1 L'application ne doit pas pouvoir être exécutée sur un appareil rooté:

Lorsque l'application est installée sur un téléphone Android rooté, elle doit refuser de s'exécuter.
Sur un téléphone rooté, divers mécanismes de sécurité peuvent en effet être désactivés. Un auteur d'attaques peut alors falsifier le code ou les données de l'application à des fins frauduleuses.
Si l'application accepte de fonctionner sur un appareil rooté, elle doit alors appliquer *a minima* les trois vérifications de sécurité suivantes: obfuscation du code (T9.1), épinglage de certificats afin d'empêcher l'interception des communications à l'aide de certificats fiables (T3.3), et chiffrement des informations sensibles par l'application, y compris celles transmises par HTTPS (T5.3).

2.7 M9 Rétro-ingénierie

T9.1 Le code de l'application requiert une obfuscation:
Divers outils permettent d'analyser le code binaire de l'application afin de déterminer s'il a fait l'objet d'une obfuscation. Sinon, le code peut provisoirement être reconstitué à l'aide d'un décompilateur. Si cela fonctionne, le code ainsi reconstitué peut être analysé afin de voir s'il est intelligible.
L'obfuscation du code en rend la logique et les algorithmes beaucoup plus difficiles à comprendre et à analyser.

3 RÉSULTATS

3.1 App1

App1 permet les échanges de fonds entre utilisateurs et peut être utilisée pour effectuer des paiements sans espèces dans des magasins ou à des distributeurs. Les utilisateurs sont identifiés par leur numéro de téléphone. Il suffit de connaître le numéro de téléphone d'un utilisateur pour pouvoir lui envoyer des fonds. Les comptes sont généralement adossés à un compte bancaire. Il est également possible d'utiliser un compte prépayé, indépendant de tout compte bancaire.

3.1.1 M1: Utilisation détournée de la plate-forme

√ T1.1 Android:allowBackup est réglé sur "False" dans le manifeste.
√ T1.2 Android:debuggable n'est pas défini dans le manifeste.
√ T1.3 Android:installLocation n'est pas défini dans le manifeste.
√ T1.4 Nous n'avons pas trouvé d'autorisations Android indues dans le manifeste.

3.1.2 M2: Stockage non sécurisé des données

x T2.1 L'application sollicite l'autorisation "android.permission.WRITE_EXTERNAL_STORAGE". N.B.: Cela ne signifie pas nécessairement que l'application écrit des données sur le dispositif de stockage externe ni que, si tel est le cas, il s'agit de données sensibles.
√ T2.2 Lorsque l'application est en marche, les captures d'écran sont désactivées.

3.1.3 M3: Communications non sécurisées

√ T3.1 L'application fonctionne uniquement sur des connexions HTTPS.
√ T3.2 L'application a refusé d'établir une connexion HTTPS avec un proxy disposant d'u' certificat non fiable.
√ T3.3 L'application a refusé d'établir une connexion HTTPS avec un proxy disposant d'u' certificat fiable. Ce comportement montre que l'application a recours à l'épinglage de certificat.
√ T3.4 L'application définit une configuration de sécurité du réseau personnalisée dans son manifeste. Cette configuration désactive le trafic en texte clair:

```
<network-security-config>
        <base-config clear textTrafficPermitted="f »lse"> »
  ...
 </base-config>
</network-security-config>
```

3.1.4 M4: Authentification non sécurisée

√ T4.1 À chaque fois que l'application est lancée, une authentification par code PIN ou empreinte digitale est requise.
√ T4.2 L'application s'éteint automatiquement après un certain délai d'inactivité. L'application se ferme après un certain délai d'inactivité.

√ T4.3 Si une nouvelle empreinte digitale est ajoutée, l'application désactive l'authentification par empreintes digitales.
√ T4.4 Les demandes d'envoi d'argent ne peuvent pas être répétées. Le serveur envoie alors le message "409 Conflict" et ne traite pas la demande de transfert de fonds.

3.1.5 M5: Cryptographie insuffisante

x T5.1 L'application utilise les algorithmes de hachage MD5 et SHA-1 ainsi que le mode de chiffrement ECB, qui offrent un niveau de sécurité faible.
MD5 in file com/appdynamics/eumagent/runtime/p000private/ae.java:
MessageDigest instance = MessageDigest.getInstance("MD5");
SHA-1 in file com/App1/android/Security/SecCore/b/a.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");
ECB in file com/App1/android/Security/SecCore/b/a.java:
Cipher instance = Cipher.getInstance("AES/ECB/NoPadding");
√ T5.2 En interceptant les requêtes HTTPS de l'application à l'aide de Burp Proxy, il est possible d'identifier le serveur auquel se connecte le client. La configuration TLS du serveur a été évaluée à l'aide de Qualys SSL Labs[9]. Sa note globale est A+.
x T5.3 Lorsque l'on intercepte les requêtes HTTPS de l'application à l'aide de Burp Proxy, les requêtes du client sont chiffrées. Cependant, le montant des fonds transférés ainsi que le prénom, le nom et le numéro de téléphone des utilisateurs concernés sont en texte clair.

3.1.6 M8: Falsification de code

√ T8.1 Nous avons réussi à installer et à utiliser l'application sur un appareil rooté.

3.1.7 M9: Rétro-ingénierie

√ T9.1 Le code de l'application a fait l'objet d'une obfuscation, comme le montre la figure 1.

3.2 App2

App2 est un service utilisé pour envoyer des fonds, effectuer des paiements ou obtenir des microfinancements. App2 n'est pas reliée à un compte bancaire. Les fonds peuvent être déposés sur le compte ou en être retirés de diverses manières, par exemple via un opérateur de télécommunications ou un commerce de détail.

3.2.1 M1: Utilisation détournée de la plate-forme

√ T1.1 Android:allowBackup est réglé sur "False" dans le manifeste.
√ T1.2 Android:debuggable n'est pas défini dans le manifeste.
√ T1.3 Android:installLocation n'est pas défini dans le manifeste.
√ T1.4 Nous n'avons pas trouvé d'autorisations Android indues dans le manifeste.

3.2.2 M2: Stockage non sécurisé des données

√ T2.1 L'application sollicite l'autorisation "android.permission.WRITE_EXTERNAL_STORAGE". N.B.: Cela ne signifie pas nécessairement que l'application écrit effectivement des données sur le dispositif de stockage externe ni que, si tel est le cas, il s'agit de données sensibles.
√ T2.2 Lorsque l'application est en marche, les captures d'écran sont désactivées.

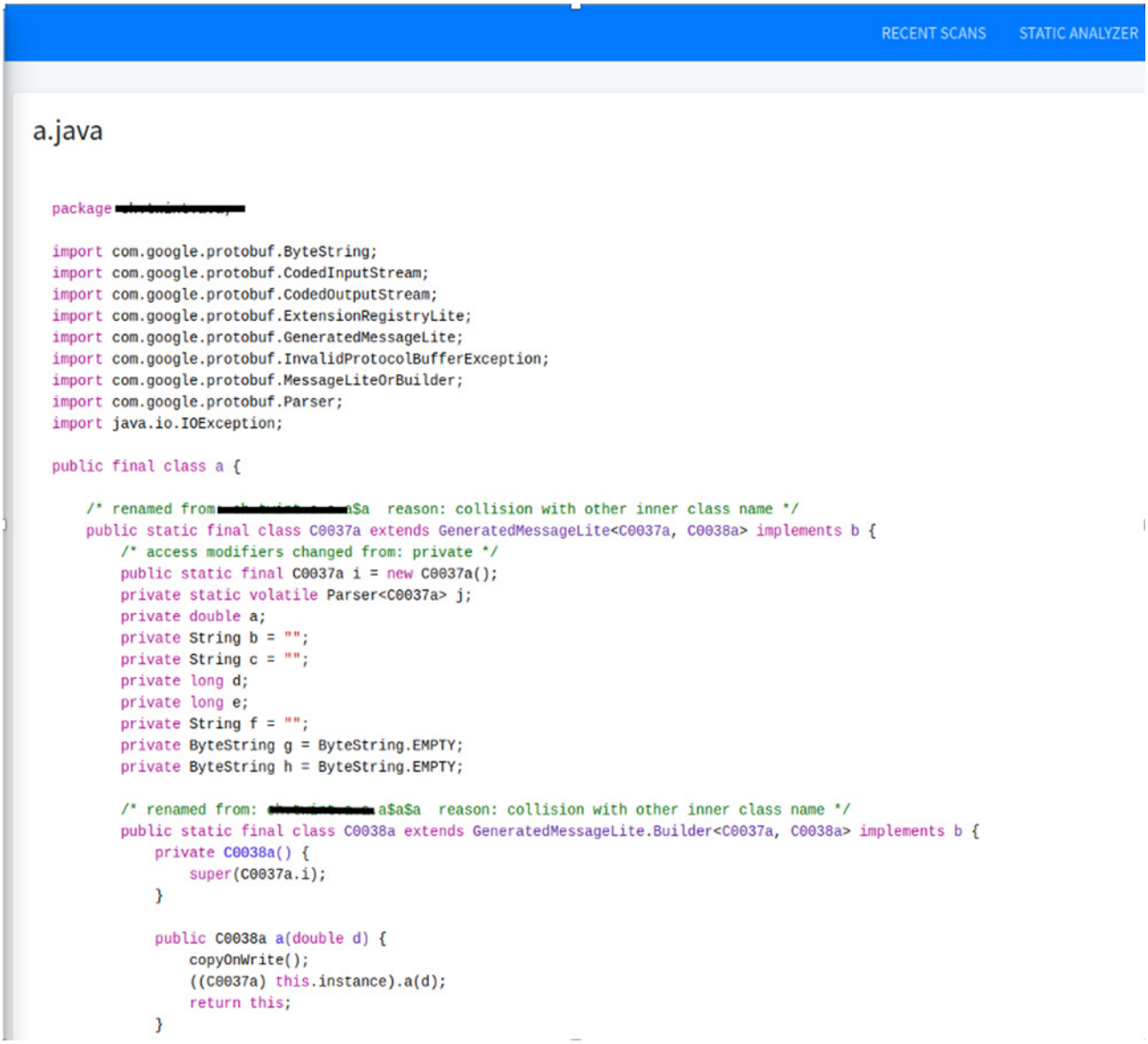
3.2.3 M3: Communications non sécurisées

√ T3.1 L'application fonctionne uniquement sur des connexions HTTPS.
√ T3.2 L'application a refusé d'établir une connexion HTTPS avec un proxy disposant d'un certificat non fiable.
√ T3.3 L'application a refusé d'établir une connexion HTTPS avec un proxy disposant d'un certificat fiable. Ce comportement montre que l'application a recours à l'épinglage de certificat.
x T3.4 Android:usesCleartextTraffic est défini comme "true" (vrai) dans le manifeste.

3.2.4 M4: Authentification non sécurisée

x T4.1 L'application n'exige pas de code PIN ou d'empreinte digitale à chaque utilisation. Un intrus qui a volé un appareil non verrouillé peut alors utiliser l'application. L'attaquant peut alors visualiser le code PUK du téléphone et consulter le montant sur le compte. La saisie du code PIN est toutefois nécessaire pour effectuer des transactions.

Figure 1 Les noms des fichiers, des classes et des variables ont été modifiés afin de rendre le code plus difficile à comprendre.



√ T4.2 La saisie du code PIN est exigée à chaque transaction. Cette protection est encore plus sûre qu'une désactivation après un délai d'inactivité.

√ T4.3 Si une nouvelle empreinte digitale est ajoutée, l'application désactive l'authentification par empreintes digitales.

x T4.4 Les demandes sensibles telles que les transferts de fonds ne peuvent pas être répétées.

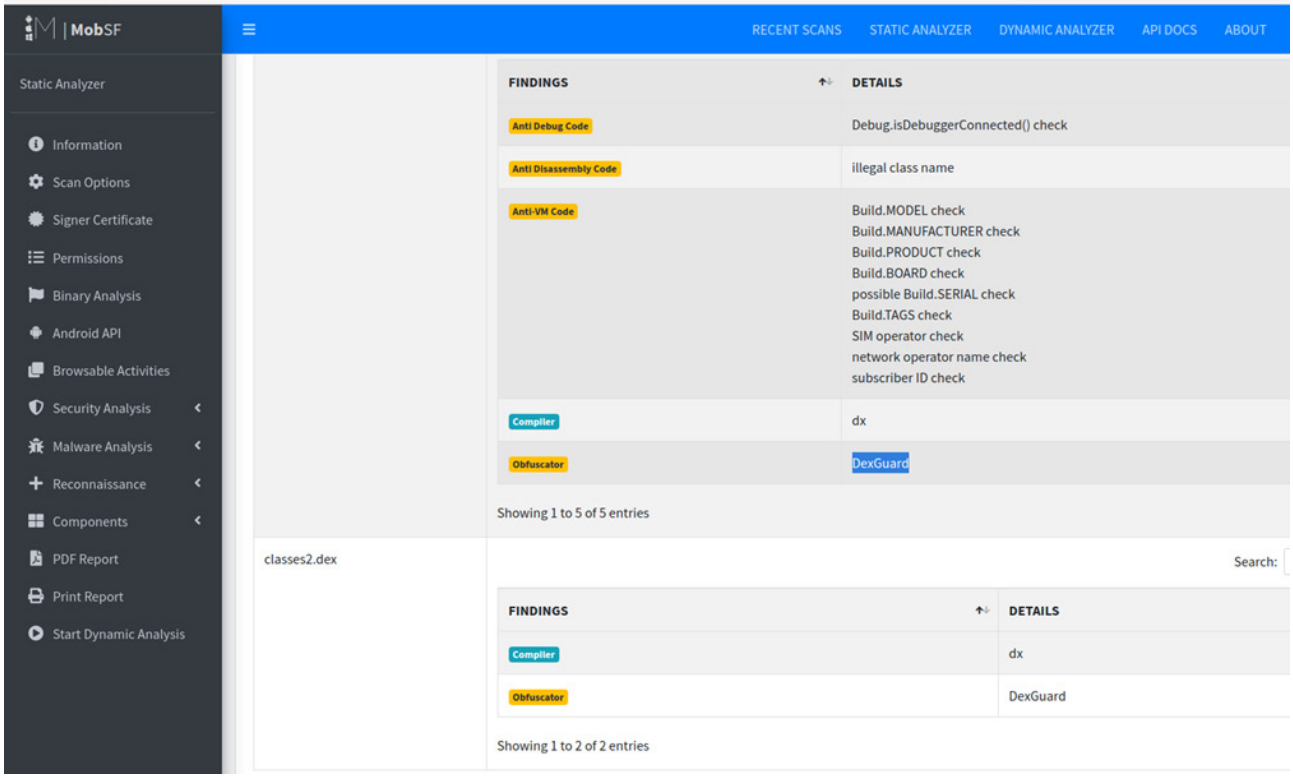
3.2.5 M5: Cryptographie insuffisante

x T5.1 L'application utilise l'algorithme SHA-1 ainsi qu'un générateur de nombres aléatoires par défaut, qui offrent tous deux un niveau de sécurité faible.
SHA-1 in file o/C1668.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");

x Random generator in file o/C1783.java:
this(juVar, d, new Random());

√ T5.2 En interceptant les requêtes HTTPS de l'application à l'aide de Burp Proxy, nous avons pu

Figure 2 App2 est protégée par DexGuard



identifier le serveur auquel se connecte l'application.
La configuration TLS du serveur a été testée à l'aide de Qualys SSL Labs[10]. Elle obtient la note globale de A+.

3.2.6 M8: Falsification de code

√ T8.1 L'application ne fonctionne pas sur les appareils Android rootés.

3.2.7 M9: Rétro-ingénierie

√ T9.1 Le code de l'application a fait l'objet d'une obfuscation par DexGuard[11] (voir figure 2).

3.3 App3

App3 est une application de paiement qui peut être utilisée pour régler des factures d'eau ou d'électricité, transférer des fonds ou effectuer des achats en ligne. Elle peut être reliée soit à un compte bancaire, soit à un portefeuille numérique associé à un numéro de téléphone.

3.3.1 M1: Utilisation détournée de la plate-forme

√ T1.1 Android:allowBackup est réglé sur "False" dans le manifeste.
√ T1.2 Android:debuggable n'est pas défini dans le manifeste.
√ T1.3 Android:installLocation n'est pas défini dans le manifeste.
√ T1.4 Nous n'avons pas trouvé d'autorisations Android indues dans le manifeste.

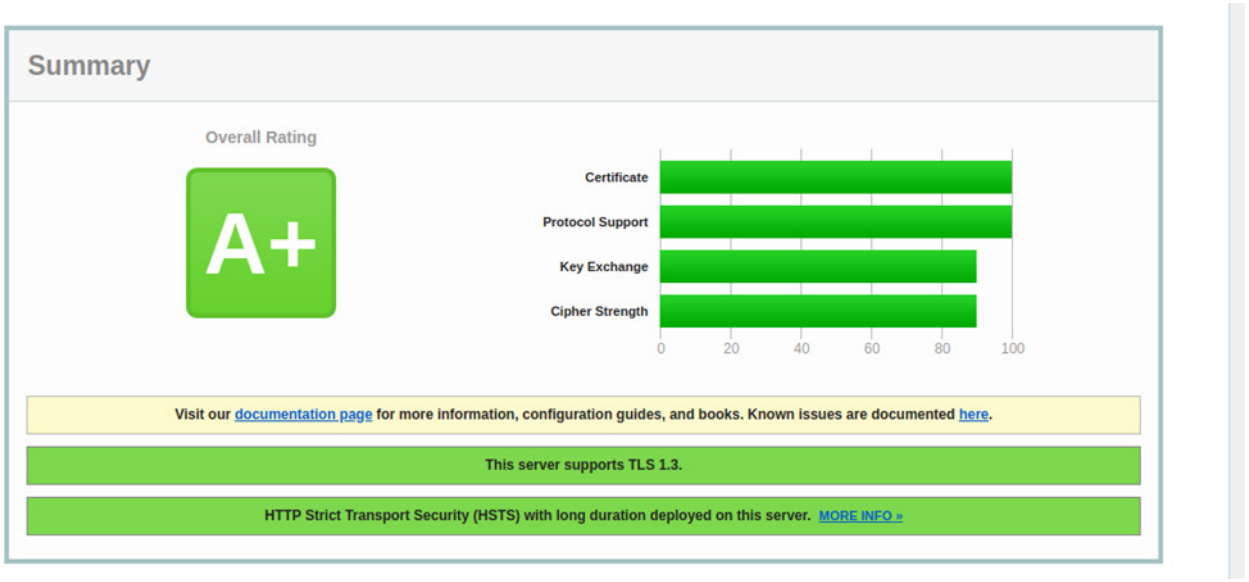
3.3.2 M2: Stockage non sécurisé des données

√ T2.1 L'application ne sollicite pas l'autorisation "android.permission.WRITE_EXTERNAL_STORAGE".
x T2.2 Lorsque l'application est en marche, les captures d'écran ne sont pas désactivées. De plus, les captures de fond d'écran depuis l'historique des tâches récentes ne sont pas floutées.

3.3.3 M3: Communications non sécurisées

√ T3.1 L'application fonctionne uniquement sur des connexions HTTPS.

Figure 3 Évaluation de la configuration SSL du serveur de App3



√ T3.2 L'application a refusé d'établir une connexion HTTPS avec un proxy disposant d'un certificat non fiable.

x T3.3 L'application accepte d'établir une connexion HTTPS avec un proxy disposant d'un certificat fiable. Ce comportement montre que l'application n'a pas recours à l'épinglage de certificat.

x T3.4 Android:usesCleartextTraffic est défini comme "true" (vrai) dans le manifeste.

3.3.4 M4: Authentification non sécurisée

x T4.1 L'application n'exige pas de code PIN ou d'empreinte digitale à chaque utilisation. Un intrus qui a volé un appareil non verrouillé peut alors utiliser l'application et consulter le montant sur le compte. La saisie du code PIN est toutefois nécessaire pour effectuer des transactions.

√ T4.2 La saisie du code PIN est exigée à chaque transaction. Cette protection est encore plus sûre qu'une désactivation après un délai d'inactivité.

√ T4.3 Si une nouvelle empreinte digitale est ajoutée, l'application désactive l'authentification par empreintes digitales.

x T4.4 Les demandes sensibles telles que les transferts de fonds ne peuvent pas être répétées.

3.3.5 M5: Cryptographie insuffisante

x T5.1 L'application utilise les algorithmes de hachage MD5 et SHA-1 ainsi qu'un générateur de nombres aléatoires par défaut, qui offrent un niveau de sécurité faible.
MD5 in file com/appsflyer/internal/ai.java:
MessageDigest instance = MessageDigest.getInstance("MD5");
SHA-1 in file u/b/a/a/o/b/j.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");
Random generator in file c/g/a/c/s.java:
Random random = new Random();

√ T5.2 En interceptant les requêtes HTTPS de l'application à l'aide de Burp Proxy, nous avons pu identifier le serveur auquel se connecte l'application.
La configuration TLS du serveur identifié a été testée à l'aide de Qualys SSL Labs[12]. Elle obtient la note globale de A+.

√ T5.3 En interceptant les requêtes HTTPS de l'application à l'aide de Burp Proxy, nous avons constaté que le corps de certaines requêtes sensibles était chiffré.
Toutefois, certaines réponses contenant des données sensibles (telles que le solde actuel du compte) ne sont ni chiffrées, ni authentifiées, et sont donc susceptibles d'être modifiées par des attaquants malveillants.

La réponse du serveur a été falsifiée durant les tests et le solde du compte de l'utilisateur s'affichant sur l'application a été modifié.

3.3.6 M8: Falsification de code

√ T8.1 L'application ne fonctionne pas sur les appareils Android rootés.

3.3.7 M9: Rétro-ingénierie

√ T9.1 Le code de l'application a fait l'objet d'une obfuscation.

4 CONCLUSIONS

Nous avons défini une méthode pour auditer les applications de DFS en déployant des efforts raisonnables. Cette méthode exclut toute analyse de la logique des applications, puisqu'une telle démarche impliquerait de pratiquer une rétro-ingénierie du code de l'application. Les trois applications testées pratiquent toutes l'obfuscation de code, ce qui rend les tentatives de rétro-ingénierie particulièrement difficiles.

4.1 Évaluation des résultats

Comme nous n'analysons pas la logique des applications, il est difficile d'estimer l'impact d'un test non réussi. Par exemple, la détection d'opérations cryptographiques non sécurisées n'implique pas nécessairement que des informations sensibles ne seront pas chiffrées de manière sûre. Un autre exemple est le fait que App1 ne chiffre pas le détail des transactions (noms, montants) réalisées via une connexion HTTPS. Puisque l'application pratique l'épinglage de certificats, le risque que les informations soient interceptées par un adversaire est pratiquement nul.
Néanmoins, tous les tests réalisés renvoient aux bonnes pratiques que sont tenues de respecter les applications financières. Les résultats présentés doivent donc être lus comme une évaluation normalisée consistant à déterminer si les applications étudiées sont conçues conformément aux bonnes pratiques. Les résultats des tests ne permettent pas, à eux seuls, de déterminer si une application est vulnérable à une attaque spécifique.

4.2 Comparaison avec le Cadre de garantie de la sécurité des DFS du Groupe de travail de la FIGI sur la sécurité, l'infrastructure et la confiance

Le Groupe de travail sur la sécurité, l'infrastructure et la confiance de la FIGI a établi un Cadre de garantie de la sécurité pour les DFS[13].
Dans son neuvième chapitre, ce cadre définit une liste de cinq catégories de bonnes pratiques. Le tableau suivant classe les 18 tests de la méthode proposée dans ces cinq catégories.

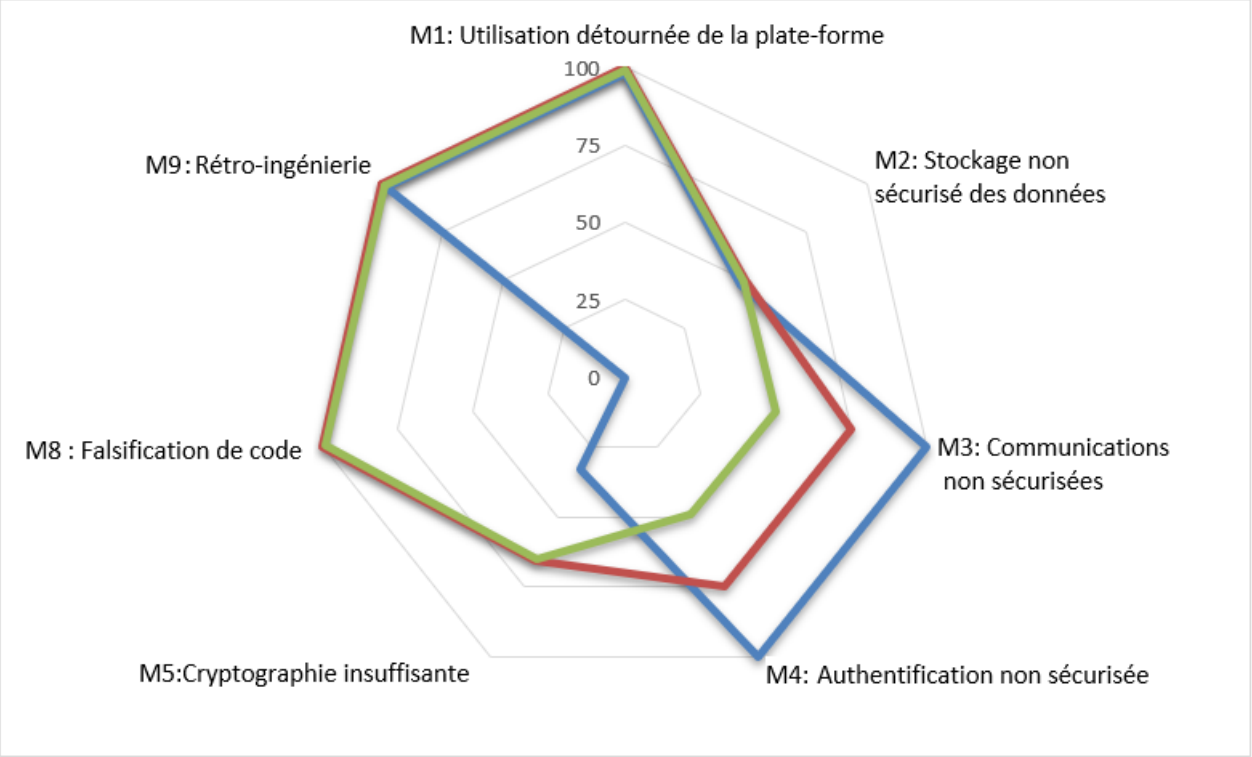
Bonnes pratiques du Cadre de garantie de la sécurité des DFS	Tests correspondants
9.1 Intégrité de l'appareil	T1.2 Android:debuggable T1.4 Autorisations dangereuses T8.1 L'application ne doit pas pouvoir être exécutée sur un appareil rooté
9.2 Sécurité des communications et gestion des certificats	T3.1 L'application doit uniquement fonctionner sur des connexions HTTPS T3.2 L'application doit détecter les attaques par intercepteur utilisant des certificats non fiables T3.3 L'application doit détecter les attaques par intercepteur utilisant des certificats fiables T3.4 Le manifeste de l'application ne doit pas autoriser le trafic en texte clair T5.1 L'application ne doit pas utiliser de primitives cryptographiques non sécurisées T5.2 Les connexions HTTPS doivent être configurées conformément aux bonnes pratiques T5.3 L'application doit chiffrer les données sensibles envoyées via des connexions HTTPS
9.3 Authentification de l'utilisateur	T4.1 Une authentification doit être requise pour accéder aux informations sensibles T4.2 L'application doit se fermer automatiquement au bout d'un certain délai d'inactivité T4.3 Si une nouvelle empreinte digitale est ajoutée, l'authentification existante par empreintes digitales doit être désactivée
9.4 Gestion sécurisée des données	T1.1 Android:allowBackup T1.3 Android:installLocation T2.1 Android.permission.WRITE_EXTERNAL_STORAGE T2.2 Désactivation des captures d'écran
9.5 Développement sécurisé de l'application	T9.1 Le code de l'application nécessite une obfuscation

4.3 Synthèse des résultats

Nous clôturons ce rapport en présentant une synthèse des résultats des tests (voir figure 4). Aucune vulnérabilité critique n'a été détectée lors des tests. Cependant, deux résultats inattendus ont été mis au jour: aucun code PIN n'est demandé pour accéder au code PUK dans l'App2, et l'App3 ne prévoit pas de chiffrement renforcé des données échangées sur connexions HTTPS, contrairement à la plupart des applications financières.

L'analyse d'applications supplémentaires permettrait d'obtenir une base de comparaison plus large et d'affiner les tests pratiqués.

Figure 4: Diagramme en étoile des résultats des tests (l'axe radial indique le pourcentage de bonnes pratiques testées que l'application met en œuvre)



Notes de fin

1 https://owasp.org

2 https://owasp.org/www-project-mobile-top-10/

3 https://developer.android.com/guide/topics/manifest/application-element#allowbackup

4 https://developer.android.com/guide/topics/manifest/application-element#debug

5 https://developer.android.com/guide/topics/manifest/manifest-element#install

6 https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_SECURE

7 Les algorithmes MD5 et SHA-1 sont utilisés pour protéger l'intégrité des données, RC4, DES, 3DES, Blowfish et ECB protègent la confidentialité, tandis que les générateurs aléatoires servent à générer des clés pour protéger l'intégrité, la confidentialité ou d'autres propriétés.

8 https://www.ssllabs.com/ssltest/

9 https://www.ssllabs.com/ssltest/

10 https://www.ssllabs.com/ssltest/

11 https://www.guardsquare.com/en/products/dexguard

12 https://www.ssllabs.com/ssltest/

13 https://www.itu.int/en/ITU-T/extcoop/figisymposium/Documents/ITU_SIT_WG_Technical%20report%20on%20Digital%20Financial%20Services%20Security%20Assurance%20Framework_f.pdf

International Telecommunication Union
Place des Nations
CH-1211 Geneva 20
Switzerland