Story 1 — Vérification de l'APK

But : recevoir et vérifier l'APK fourni par le PO afin de travailler sur une base commune.

Definition of Done (DoD) : chaque étudiant vérifie le hash avec `sha256sum app.apk` et documente le résultat.

Résultats

- **Hash SHA256 de l'APK** :

`F01F08C8B6E2FE4612E81BC7E3A3BA9440DAD0D7A962F4E67640390DD721D528`

- **Commandes utilisées (exemples)** :
  - PowerShell : `Get-FileHash -Path "PATH" -Algorithm SHA256`
  - Linux/macOS : `sha256sum [filename]`

Story 2 — Permissions dangereuses & vulnérabilités critiques

Permissions identifiées

- `android.permission.READ_CONTACTS` — **Dangereux**
  - **Risque** : exfiltration du carnet d'adresses, atteinte à la confidentialité.
- `android.permission.WRITE_EXTERNAL_STORAGE` — **Dangereux**
  - **Risque** : modification/suppression/lecture de fichiers externes, atteinte à l'intégrité des données.

Vulnérabilités critiques identifiées

1. **Compatibilité avec versions Android non sécurisées**
   a. **Détails** : `minSdk=19` (Android 4.4.x). Ces versions sont hors support et contiennent de nombreuses vulnérabilités.
   b. **Recommandation** : cibler au minimum Android 10 (API 29) si possible.
2. **Trafic en clair activé** (`android:usesCleartextTraffic=true`) — **Critique**

    a. **Risque** : autorise HTTP en clair, FTP, etc. Exposition à interception / modification des communications (MITM).
    b. **Recommandation** : désactiver `usesCleartextTraffic` ou configurer Network Security Config pour autoriser uniquement des hôtes de confiance en HTTPS.
3. **Utilisation d'un mode de chiffrement vulnérable (CBC + PKCS5/PKCS7) — Critique/Élevé**
    a. **Fichiers affectés** :
        i. `com/oblador/keychain/cipherStorage/CipherStorageKeystoreAESCBC.java`
        ii. `com/pushwoosh/internal/a/a.java`
    b. **Risque** : vulnérable aux attaques par padding oracle. Recommandation : utiliser AES-GCM (AEAD) ou gérer l'authenticité avec HMAC+AES-CBC (si impossibilité de migrer immédiatement).
    c. **Références** : OWASP Top10 M5, MASVS MSTG-CRYPTO-3, CWE-649.

Story 3 — Extraction de fichiers (classes.dex & AndroidManifest.xml)

But : extraire `classes.dex` et `AndroidManifest.xml` pour préparer la décompilation.

DoD : livrer `classes.dex` et `AndroidManifest.xml` avec leur hash SHA256.

Résultats

- **Hash SHA256 de `classes.dex`** :

`57898C5978715DA96560ED2692B4A8FA1A8E914B37988DCF369CC59951F6C429`

- **Hash SHA256 de `AndroidManifest.xml`** :

`80B9B101E3BF02F03EEC4F2C9C9CCF4D61403612E1435949B51CEC8B85D7A276`

- **Méthode** : renommer `app.apk` → `.zip` ou utiliser `unzip` / `jar xf`, puis `Get-FileHash classes.dex -Algorithm SHA256`.

Story 4 — Décompilation avec Apktool

Commande utilisée :

```
apktool d nom_apk -o apk_decompiler
```

Observations : apktool a permis d'obtenir l'arborescence des ressources (`res/`), fichiers XML décodés, manifest et Smali/ressources nécessaires à l'analyse.

(Capture d'écran et sortie détaillée peuvent être ajoutées en annexe si nécessaire.)

Story 5 — Analyse de `strings.xml`

DoD : `strings_findings.md` avec extraits (≤25 mots) et risques associés.

Secrets critiques détectés (apktool)

Fichier : `apktool_output/res/values/strings.xml`

1. **Google API Key — CRITIQUE**
    a. **Extrait (15 mots)** : `<string name="google_api_key">AIzaSyBTgztvImsUfMWDa41PCrDWAj7dmyIDhUg</string>`
    b. **Risque** : accès non autorisé aux services Google Cloud, vol de données.
    c. **CVSS estimé** : 9.1
2. **OAuth Client ID — CRITIQUE**
    a. **Extrait (10 mots)** : `<string name="default_web_client_id">717748501407-v621tmfvqd7etdouc3df5vuv31scle0g.apps.googleusercontent.com</string>`
    b. **Risque** : détournement d'authentification OAuth / usurpation.
    c. **CVSS estimé** : 8.5
3. **Firebase Database URL — ÉLEVÉ**

    a. **Extrait (8 mots)** : `<string name="firebase_database_url">https://application-client-nickel.firebaseio.com</string>`
    b. **Risque** : énumération / accès potentiel à la base.
    c. **CVSS estimé** : 7.8
4. **API Banking Endpoint — MOYEN**
    a. **Extrait (7 mots)** : `<string name="base_url_banking">https://api.nickel.eu/customer-banking-api</string>`
    b. **Risque** : reconnaissance de l'infrastructure.
    c. **CVSS estimé** : 5.2
5. **API Auth Endpoint — MOYEN**
    a. **Extrait (7 mots)** : `<string name="base_url_authentication">https://api.nickel.eu/customer-authentication-api</string>`
    b. **Risque** : reconnaissance / attaques ciblées.
    c. **CVSS estimé** : 5.2

Autres informations sensibles

- **Facebook App ID** : `<string name="facebook_app_id">1659302534172516</string>` — risque d'utilisation non autorisée d'intégration sociale.
- **Pushwoosh AppID** : `<string name="pushwoosh_appid">7A240-86F24</string>` — configuration push exposée.

Recommandations générales :

- Ne jamais stocker clés/API/OAuth secrets dans `strings.xml` ou code embarqué.
- Utiliser des secrets stockés côté serveur ou via un mécanisme sécurisé (Android Keystore pour secrets locaux, remote config / secrets manager côté backend).

Story 6 — Décompilation & analyse du bytecode

DoD : `code_findings.md` avec explication pour chaque extrait.

Vulnérabilité 1 — CBC + PKCS5/PKCS7

- **Gravité** : Élevée
- **Description** : utilisation de AES-CBC avec PKCS5/7 exposant à padding oracle.
- **Fichiers concernés** :
  - `com/oblador/keychain/cipherStorage/CipherStorageKeystoreAESCBC.java`
  - `com/pushwoosh/internal/a/a.java`
- **Impact** : compromission possible des données chiffrées (jetons, mots de passe).
- **Remédiation** : migrer vers AES-GCM (mode AEAD) et s'assurer d'une gestion correcte des IV/nonce.

Vulnérabilité 2 — Secrets codés en dur

- **Gravité** : Moyenne (impact potentiellement élevé)
- **Fichiers concernés** :
  - `com/bumptech/glide/load/Option.java`
  - `com/bumptech/glide/load/engine/DataCacheKey.java`
  - `com/bumptech/glide/load/engine/EngineResource.java`
  - `com/bumptech/glide/load/engine/ResourceCacheKey.java`
  - `com/pushwoosh/reactnativeplugin/InboxUiStyleManager.java`
- **Remédiation** : supprimer secrets du code, mettre en place configuration côté serveur et variables d'environnement; rotation immédiate des clés compromettues (Google API Key, OAuth client secrets, etc.).

Vulnérabilité 3 — Injection SQL & stockage non sécurisé

- **Gravité** : Élevée
- **Fichiers concernés** :
  - `com/pushwoosh/inapp/f/b.java`
  - `com/pushwoosh/inbox/e/b/b.java`
  - `com/pushwoosh/internal/network/f.java`
  - `com/pushwoosh/repository/LockScreenMediaStorageImpl.java`
  - `com/pushwoosh/repository/PushBundleStorageImpl.java`
  - `com/pushwoosh/repository/c.java`
- **Description** : requêtes SQL construites par concaténation, absence de paramétrage ; stockage de données sensibles en clair.

- **Impact** : exposition / modification des données, élévation de privilèges.
- **Remédiation** :
    - Utiliser des requêtes paramétrées / ORM.
    - Chiffrer les données sensibles au repos (p.ex. via Android Keystore + chiffrement des champs).

Story 7 — Recompilation de l'APK

But : recompiler l'APK avec apktool pour vérifier la réversibilité.

DoD : fournir le APK recompilé et le logo de build.

Commande utilisée

```
apktool b "chemin/vers/dossier_décompilé" -o
"chemin/vers/app_recompile.apk"
```

Log (extrait)

```
I: Using Apktool 2.12.1 on app.apk with 8 threads
I: Checking whether sources have changed...
I: Checking whether resources have changed...
I: Building resources with aapt2...
I: Building apk file...
I: Importing assets...
I: Importing lib...
I: Importing unknown files...
I: Built apk into: app_recompile
```

Story 8 — Signature de l'APK

Création d'un keystore

```
keytool -genkey -v -keystore mon_keystore.keystore -alias mon_alias
-keyalg RSA -keysize 2048 -validity 10000
```

Signature de l'apk

```
apksigner sign --ks mon_keystore.keystore --ks-key-alias mon_alias
mon_application.apk
```

Vérification de la signature

```
jarsigner -verify -verbose -certs mon_application.apk
```

Story 9 — Vérification Firebase

But : tester l'accès public à la base Firebase découverte dans `strings.xml`.

URL testée : https://application-client-nickel.firebaseio.com

Résumé des tests

- **Résultat global** : **SÉCURISÉ** (accès protégé)
- **Risque de fuite** : **FAIBLE** (accès public refusé)

Tests réalisés (extraits)

1. `GET /.json` → réponse :

```
{ "error": "Permission denied" }
```

2. Énumération d'endpoints (`/.json`, `/users.json`, `/config.json`, `/public.json`, `/test.json`) → toutes retournent `401 Unauthorized` / `Permission denied`.
3. Test méthodes HTTP (GET, POST, PUT, DELETE) → réponses : `Unauthorized` ou `BadRequest` selon l'endpoint/méthode.
4. `?shallow=true` → `Unauthorized`.

Conclusion : la base Firebase nécessite authentification et ne permet pas l'accès en lecture publique.